Application Of Neural Network On PLC-based Automation Systems For Better Fault Tolerance And Error Detection

By

Bhargav Joshi

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Computer Science and Software Engineering

Auburn, Alabama
August 3, 2019

Keywords: PLC, neural, network,
automation, ladder, python

Approved by

David Umphress, Chair, Professor of CSSE
Bo Liu, Assistant professor of CSSE
Anh Nguyen, Assistant professor of CSSE

ABSTRACT

Neural networks have a wide range of applications such as building complex equations using the input and output characteristics of functions, predictions of outputs, error detections, monitoring complex systems, etc. Neural network's capabilities of monitoring the system, error detection, and predictions merged with Programmable Logic Controllers (PLC) can improve the fault tolerance and error detections in automation systems. While the PLC program is being tested in the simulated environment before it is implemented in the automation system, the values of PLC's I/O ports, timers and critical variables during the execution of the program can be used to train a neural network and prepare it to monitor the system. Execution of the trained neural network in parallel with the PLC's execution where the inputs and the outputs to the PLC are also supplied to the trained neural network, adds an artificial intelligence inspired system monitor. A neural network based system monitor learns the characteristics of the automation system using PLC's port values and internal variables during the training. A successfully trained neural network can detect a malfunction or abnormal behavior in the automation system when the outputs to the automation system generated by the PLC and the outputs generated by the neural network are compared. The abnormal behavior of an automated system could have been caused by intrusions in which the PLC code has been altered by external entities, hardware faults, malfunctions on PLC's I/O ports, mishandling of the system by the operators, etc. Addition of AI-based monitor to the automation system provides an additional layer of security and helps the system run efficiently since neural network's prediction capability can alert the operators if abnormal behavior in the system starts to take place before it is too late to recover.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. CHAPTER-1: PROBLEM DESCRIPTION

PLC, also known as Programmable Logic Controllers, are referred to as the heart of industrial automation. They operate substantial automation systems such as car assembly lines, manufacturing plants, high voltage distribution systems, automated boilers, etc., all of which are complicated systems where a malfunction can have a significant financial impact or put lives at risk.

This places the onus on system designers and programmers to program PLCs so that they not only perform useful functions but also detect if an error has occurred and recover if possible. This has always been a difficult task due to having to work with unique or specialized devices that are not equipped with feedback sensors capable of reporting a failure, relegating to supervisory controllers the job of detecting problems. The difficulty of the task has been heightened as industrial automation systems have become networked. Instead of just dealing with the mechanical and process soundness of systems, developers have to take into account how resilient the system is to malicious cyber activity. It is not hard to imagine the damage that could result if a PLC in a dairy that controls a large boiler used to homogenize milk had its temperature parameters changed to the point where the pressure caused an explosion. Such an event could be caused by a malfunctioning temperature sensor, faulty pressure sensor, human error, or a cyber exploit.

While computing platforms were once too immature to detect similar scenarios in the past, improvements in hardware and software now offer possible solutions. For example, so-

called do-it-yourself hardware, such as Arduino and Raspberry Pi, put forward the notion of ultra-low-cost computing and connectivity. Artificial intelligence, too, has advanced to the stage where it is possible to observe the behavior of a mechanical system, learn what constitutes "normal" operation, and alert when functions exceed a certain threshold.

The solution proposed here is to employ neural networks to monitor the behavior of off-the-shelf PLCs with the purpose of detecting abnormal behavior. The neural network observes PLC input and output over time in order to develop a model that predicts what output should occur given an input. Actual outputs that fall outside a threshold of predicted outputs signals abnormal behavior that might be caused by a hardware malfunction or cyber intrusion.

The specific goals of this research are the following:

1. Determine how to gather training data from an already programmed PLC.
2. Identify the correct neural network architecture and training algorithm that will predict outputs from inputs.
3. Employing the prediction model to discriminate normal from abnormal PLC operations.

2. CHAPTER-2: PREVIOUS WORK

This chapter surveys previous work done on making automation systems more fault tolerant and more secure. Additionally, chapter 2 explores how the concept of machine learning can be used to improve the system as a whole.

2.1 PROTOCOLS:

Leading automation companies such as Siemens, Allen Bradley, and Honeywell have developed secure protocols to prevent communication breaches in automated systems. For example, Siemens PLCs securely communicate through the PROFIBUS protocol, is based on recognized international standards. The protocol architecture uses the OSI (Open System Interconnection) reference model in which the second Layer of the OSI reference model aids this protocol to regulate the data security and data transmission in PROFIBUS networks [6] [13].

These protocols are made only for communication among PLCs and any network-related devices attached to them. They can detect an error in communication through error detecting codes employed in the protocol. However, they cannot detect if the error is caused by corrupted data from either malfunction or intentional breaches. Constantly monitoring the data being sent to and received by the PLC would provide better reliability and security. It is impossible to verify the correctness of the data using only errors. Correctness requires knowledge of the context in which the data is used. Individual data elements may be free of errors, but they may not produce the intended result when taken together.

2.2 SIGNAL FEEDBACKS FROM EDGE DEVICES:

Automation systems are made of edge devices such as motors, sensors, actuators, etc., controlled by programmable logic controllers. Malfunctions in edge devices or PLCs can occur for various reasons. Since protocols fail to detect some of those malfunctions, system designers implement feedback signals on edge devices to determine if the device is either in the working state or not [8]. These feedback signals are usually a one-bit line that connects an edge device to a PLC's I/O port. If the edge device stops working for any reason, the value of the feedback signal from the edge device is automatically inverted, to signal a change in state. These feedback signals are interlocked within the PLC code to detect if any of the edge devices have stopped working.

This solution still does not solve the problem of verification of correctness of the data received from edge devices. Also, not all hardware can support a feedback system. Feedback-supported edge devices increase cost in the manufacturing of an automation system, possibly rendering it beyond the reach of customer budgets.

2.3 HUMAN MACHINE INTERFACE (HMI):

HMI also is known as Human Machine Interface is a live graphical representation of an automated system with added controls for humans to access it. It usually consists of a live feed of the data and parameters from edge devices, together with controls to change or handle the parameters of the system manually. The purpose of an HMI is to monitor the automated system and provide controls for it to change the settings, turn on/off connected devices, force start certain parts of the system, emergency stop, manual override, etc. These features add an additional layer of security to the system since the human element can be a part of the system.

Figure-1 illustrates an HMI of an automated turbine system. The HMI displays the core temperature of the turbine, Rotations per minute, a live feed of the oil cooling system and vibrations on the turbine shaft. Maintenance engineers can monitor the system for abnormal and possibly shut down the system in case of an emergency.

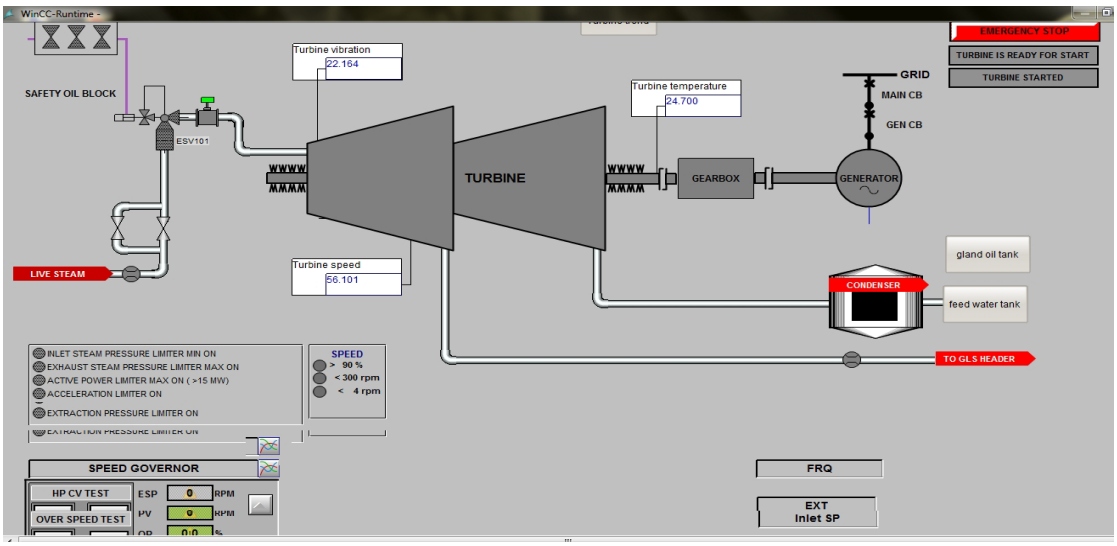


Figure 1: HMI of an automated turbine system (example)[10]

The drawback of HMI is that humans do the monitoring part of the system, leading to the possibility of human error. Someone with malicious intent gaining unauthorized access to the HMI can manipulate the system using the access provided on the HMI. These HMIs are connected to the PLC as network devices, making it vulnerable to distant hacks.

2.4 MACHINE LEARNING:

Given the unprecedented availability of data and computing resources, there is widespread renewed interest in applying data-driven machine learning methods to problems for which the development of conventional engineering solutions is challenged by modeling or algorithmic deficiencies [21].

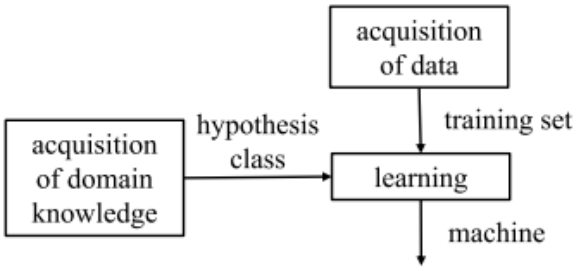


Figure 2: Machine Learning methodology [21]

Machine learning is an efficient tool to predict the output from a set of inputs. A data set made of several inputs and corresponding outputs provides to the machine learning code, and the program learns how inputs are related to outputs during the training process. The training process is done through machine learning algorithms (refer to Fig-2). Once the training process is finished, the code can predict the output from the given input without having to manually figure out the relation between inputs and outputs. One of the popular approaches to machine learning is neural networking.

2.4.1 ARTIFICIAL NEURAL NETWORKS(ANN):

Artificial neural networks are computational systems inspired by the biological neuron's network inside an animal brain. "The recipe for understanding natural intelligence and achieving strong AI is simple. If we can construct synthetic brains that mimic the adaptive behavior displayed by biological brains in all its splendor, then our mission has succeeded" [12]. A neural network is not an algorithm but a framework for many different machine learning algorithms to work together and process large data sets or data inputs [2]. It models connections of neurons in a brain where each neuron is a node connected to other nodes.

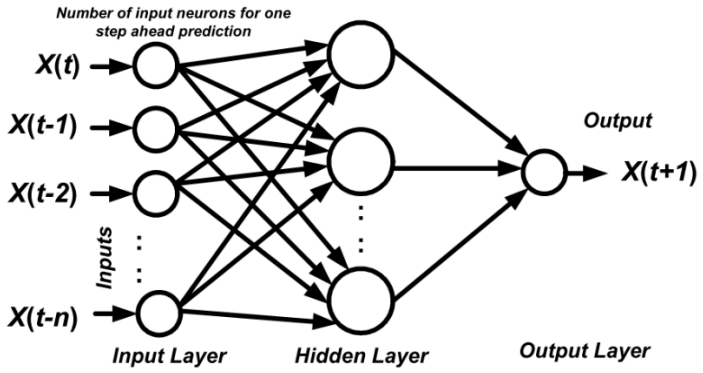


Figure 3: Number of input neurons for one-step-ahead forecasting [19]

Figure-3 represents an example of a neural network structure to predict an output from multiple inputs. Inputs pass through the web of neurons where every connection within the network has a weighted multiplier. The desired output is generated through the series of arithmetic operations done on inputs based on multipliers on every connection with neurons [19].

There are various algorithms to calculate weighted multipliers on the connections between neurons. The algorithm generates values of weighted multipliers using the provided set of expected output and input [19]. This process is called the training of neural networks. Once the training process is finished, the trained neural network is ready to predict the output based on inputs. This trained neural network can be deployed and run actively to generate outputs dynamically from provided inputs. Gradient descent, Newton's method, Conjugate Scaled Gradient, Quasi-Newton method, Levenberg-Marquardt are a few examples of popular training algorithms for neural networks [29].

2.5 MERGING NEURAL NETWORKS TO PLC OPERATIONS:

A PLC can be envisioned as a box that accepts various inputs from edge devices such as sensors and switches. It transforms the inputs into output signals that control edge devices such as motors, conveyor belts, solenoid valves, etc. Briefly, it can be said that there is a set of inputs and a set of outputs, and are related to each other through a box that is the PLC. This scenario can be compared to a neural network where there is a set of inputs and a set of outputs that are related to each other through a box, which is the neural network. Neural networks cannot re-do the functionality of the PLC; instead, they predict the chances of an output resulting from a set of specific inputs.

The neural network runs in parallel to the PLC. Both get the same inputs. The PLC's output deviating from what is predicted by the neural network signifies the possibility of an edge device generating corrupt data [18].

Employing a neural network in parallel with PLC operation solves the problem of verifying the correctness of the data coming from edge devices or the data generated by PLC since the network learns how the automation system on the PLC operates. It also eliminates the need to monitoring the system manually using HMI, thus eliminating the possibility of human error.

3. CHAPTER-3: SOLUTION

The objectives of this measure are as follows:

1. Determine how to gather training data from an already programmed PLC.
2. Identify the correct neural network architecture and training algorithm that will predict outputs from inputs.
3. Employing the prediction model to discriminate normal from abnormal PLC operations.

This chapter describes solutions to these problems.

3.1 THE CONCEPTUAL DESIGN:

As described in chapter 2, PLCs and neural networks can be visualized as black boxes that accept various inputs and generate appropriate outputs. Implementation of a neural network integrated PLC (NNPLC) was done as follows:

1. A PLC program for an automation system was built as per provided system requirements using a PLC and needed edge devices.
2. A separate machine was prepared with proper connections made with the automation system, and a neural network core (NN core) was implemented in it with fitting parameters.
3. The neural network core was put in training mode, and the automation system's was simulated in the lab environment. This step is divided into two phases.

a. Gathering of training data

b. Feeding training data to the NN core while it was in training mode

4. Upon achieving the acceptable training error, the trained NN core was deployed and executed in parallel with the PLC.
5. The PLC generated outputs and the neural network core generated outputs were compared, and the difference was computed to see if there were any errors.

In step 5, appropriate statistical model to identify actual errors was implemented to differentiate between the errors caused by abnormal behavior of the PLC program and the errors caused due to the leftover training error during the training of the neural network core.

3.1.1 NNPLC IN TRAINING CONFIGURATION:

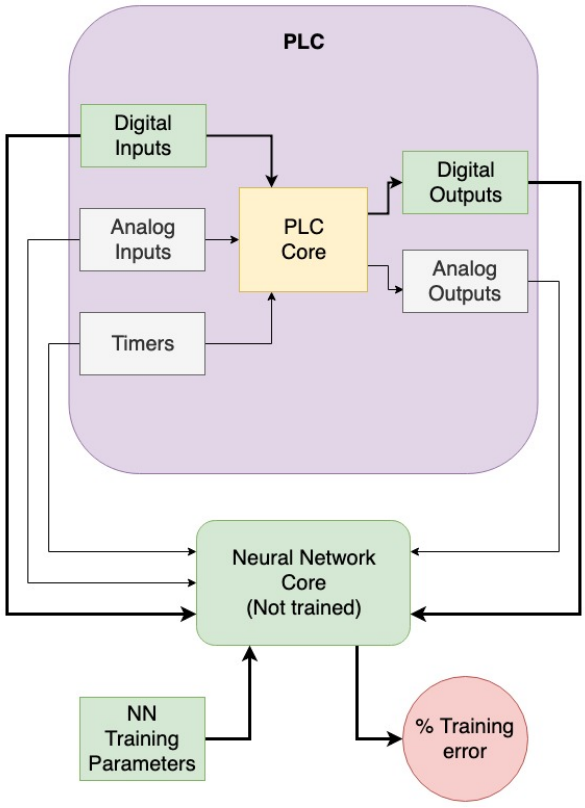


Figure 4: Block diagram of training configuration

Fig-4 shows the training configuration of NNPLC. As per this block diagram, the PLC was built with digital I/O modules, analog I/O modules, and timers. Timers are usually part of the PLC core but are shown as separate components to explain the training configuration. PLC core's job is to execute the program.

In the experiment, an untrained neural network core was implemented with the feedforward network architecture and a training algorithm. The untrained neural network core gathered data from modules while running the simulation of the PLC code and created a data set for the training algorithm. After enough training data was gathered from normal PLC operation, the training algorithm started training the network with provided parameters and generated data set, and generated the final training error. The neural network core was ready to be deployed as soon as the acceptable training error was achieved.

3.1.2 NNPLC IN DEPLOYED CONFIGURATION:

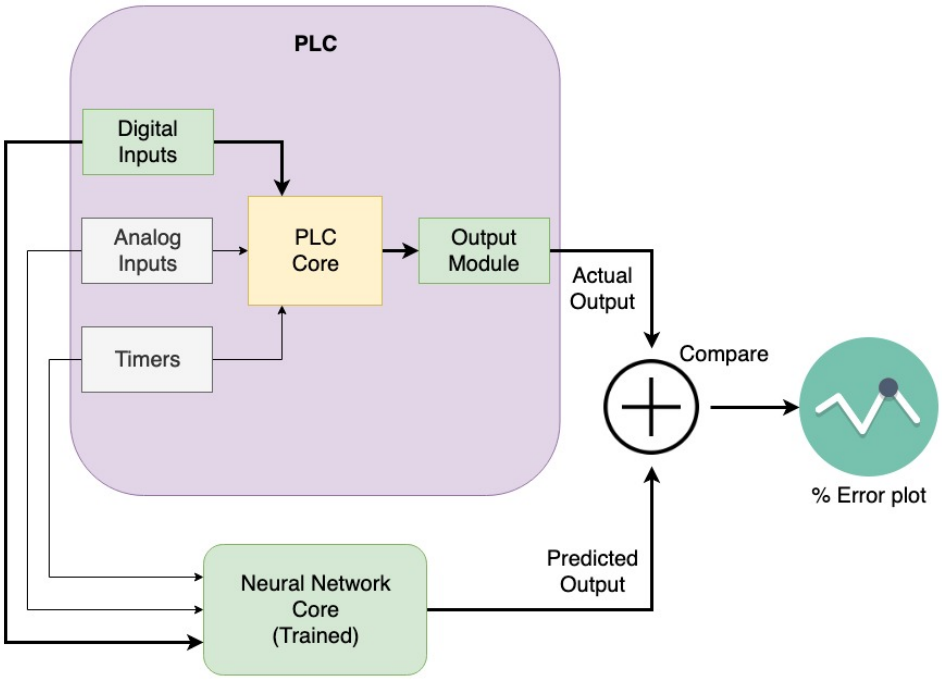


Figure 5: Block diagram of the deployed configuration

Fig-5 shows the block diagram of NNPLC configuration in its deployed mode. As shown in fig-5, PLC core and NN core receive the same inputs from PLC's timers, analog input module, and digital input module. Since the NN core was trained during the simulation of the PLC program, both PLC and NN core should ideally generate similar outputs. Here it was made sure that the NN core had the least possible training error. The difference between the PLC generated outputs (actual outputs) and the NN core generated outputs (predicted outputs) were calculated through a mathematical comparator. The percentage error plot produced by the comparator showed a significant difference in outputs produced by both PLC and NN core if the PLC started producing abnormal outputs due to any reasons (i.e., malfunction of edge devices, intrusions in the system, injection of malware or viruses, modification of the PLC program, etc.). This result happened because the NN core predicted outputs that were the same as how the normal operation of the PLC would generate, considering that it was trained with the data gathered during the normal operation of the PLC.

3.2 TACKLING KEY CHALLENGES TO IMPLEMENT THE CONCEPT DESIGN:

To implement the concept design of NNPLC, there were three major challenges that needed to be tackled. Solutions to these problems required fundamental knowledge of electrical engineering, neural networking, and software engineering.

3.3 CHALLENGE#1:

Problem description: Determine how to gather training data from an already programmed PLC.

An important question in neuroscience is how neural representations of sensory input are functionally organized. Güçlü and Van Gerven show that neural responses to sensory input

can be modeled using recurrent neural networks that can be trained end-to-end [7] [12]. For any neural network to work, it requires data to be trained. In our case, the training data was the set of inputs given to the PLC and the set of outputs produced by the PLC. The training data set for NN core was obtained from PLC's visible parts, i.e., external digital and analog ports. This data was organized in such a way that values read on the ports were time synchronized. The capture time interval to read the data from PLC's ports was calculated using the following relation.

$$\boldsymbol{Capture\ Time\ Interval = PLC\ Cycle\ Time \times Integer\ Multiplier\ n}$$

To understand this, let's consider an example of a PLC with the cycle time of 50 microseconds; meaning that PLC scans the entire ladder logic and updates values of program's variables and corresponding hardware ports every 50 microseconds. The data on the physical ports must be captured every 50 microseconds times an integer multiplier. This multiplier could be in the range from 1 to infinity. The ideal value of this multiplier should be 1 to prevent the missing of the capture of certain PLC program states. It should neither be less than one nor a non-integer value because doing so would introduce faulty captures in the training data set, meaning captures containing updated input values as well as incompletely processed output values. Feeding faulty captures would ostensibly train neural networks to produce faulty outputs. Time synchronization during the capture is one of the critical factors to gather the training data from PLC. The primary purpose of integer multiplier was to adjust the capture rate. It could be increased from one to any higher integer value in order to reduce the capture rate depending on the characteristic of the PLC program.

3.3.1 CONTENTS OF TRAINING DATA SET:

Most PLC programs handle physical hardware connected to its ports by processing inputs collected mainly from sensors and input devices such as switches or latches. Since the inputs and

the outputs to a PLC are digital and analog values, the training data set should contain digital and analog values. If the PLC uses built-in timers in its logic, then their dynamic values should also be captured and recorded in the training data set to include time dependency in the relation between inputs and outputs. The format of the training data set should take the following format.

INPUTS			**OUTPUTS**	
Timer(s)	**Digital In**	**Analog In**	**Digital Out**	**Analog Out**
300 milliseconds	0 1 1 0 1 1 1 1	3.9 V	1 1 0 0 0 0 0 1	21.5 V
400 milliseconds	0 0 0 1 0 0 0 0	1.4 V	1 0 0 0 1 0 1 1	5.2 V

Table 1: Format of training data set (example)

Table-1 shows an example of a training data set. There could be multiple timers and more digital and analog values in the real system. It should be modified based on the number of IO available on PLC ports and the number of timers used. Analog and digital values come from the capture done on ports. Timer values are not available physically on ports but can be read by software integration with PLC's programming software.

Why not collect analog and digital values through software integration? If it is not done externally then, in case of damaged/interrupted output lines to an edge device, faulty values may be physically present on the port. The NN core will not detect them since it would be reading the internal values of port variables instead of the actual values on ports. To build this data set, we had to harvest digital values, analog values, and timer values from PLC.

3.3.2 HARVESTING DIGITAL VALUES:

Digital values on the PLC's I/O ports can be viewed as parallel values. The shift registers were used to convert parallel values to a serial value to harvest all of the digital values from ports. These values were stored in the training data set.

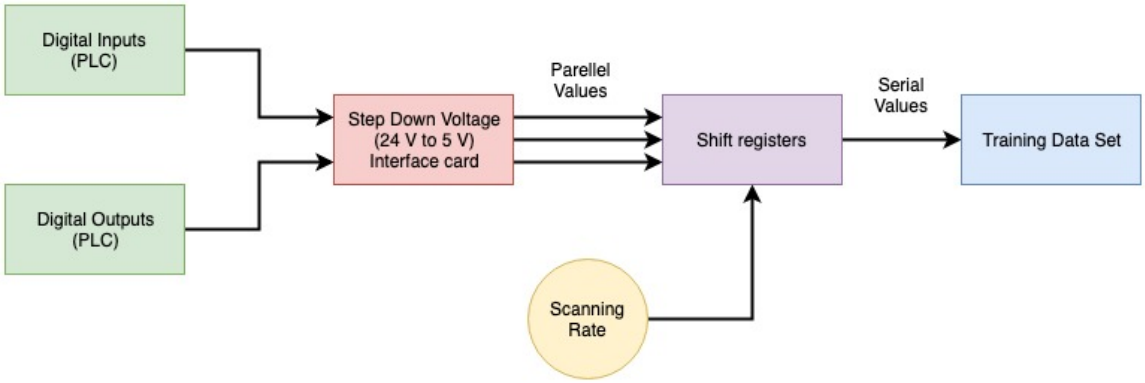


Figure 6: Harvest Digital IO values using shift registers

Fig-6 shows a data flow diagram on how to harvest digital I/O with shift registers. As shown in the diagram, digital values on PLC ports go through two stages, the step-down voltage converter and the shift registers, to store them into the training data set.

Generally, PLCs operate on 24 V. Most of the shift registers available in the market are made to work with low power circuits that usually operate on 5 V and 3.3 V. The first step to harvest digital I/O from PLC is to design an interface card between PLC and shift registers. The interface card is made of DC-DC buck converters [22]. Digital systems have two states, '1' and '0'. PLCs consider 24 V signal as the digital state 1, and 0 V signal as the digital state 0. The DC-DC buck converter can step down any voltage signal greater than 24 V signal to 5 V signal and any voltage signal less than 5 V signal to 0 V signal with proper configuration. At the output side of the interface card represents the digital state '1' as the 5 V signal and the digital state '0' as the 0 V signal. We used LM2596 buck converters to design the interface card.

Fig-7 shows LM2596 buck converter and its voltage adjustment instructions. We adjusted each buck converter to transform 24 V signal to 5 V signal. We dedicated a configured converter to each digital I/O. Once the interface card was configured, it was used as the communication bridge between the PLC's I/O port and shift registers.

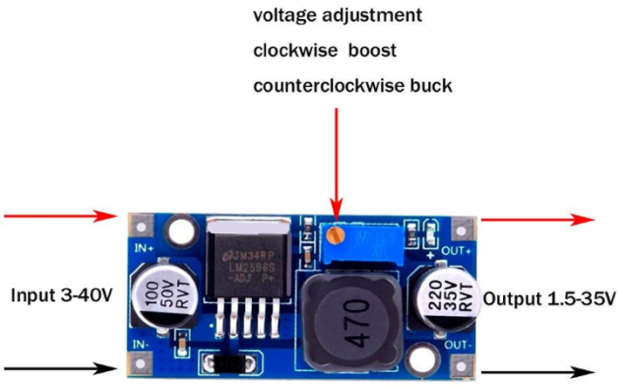


Figure 7: E-boot LM2596 buck converter [30]

The internal structure of a shift register is consisted of a chain of flip-flops in cascade, with the output of one flip-flop connected to the input of the next flip-flop. All flip-flops receive common clock pulses, which activate the shift of data from one stage to the next" [15]. This structure of flip-flops in the shift registers accepts the parallel load and shifts it into the serial load. Therefore, we used shift registers to format parallel data in such a way that the neural network's training algorithm could process it. We used Texas instrument's SN74HC165 shift registers. The HC165 devices are 8-bit parallel-load shift registers that, when clocked, shift the data toward a serial output [23].

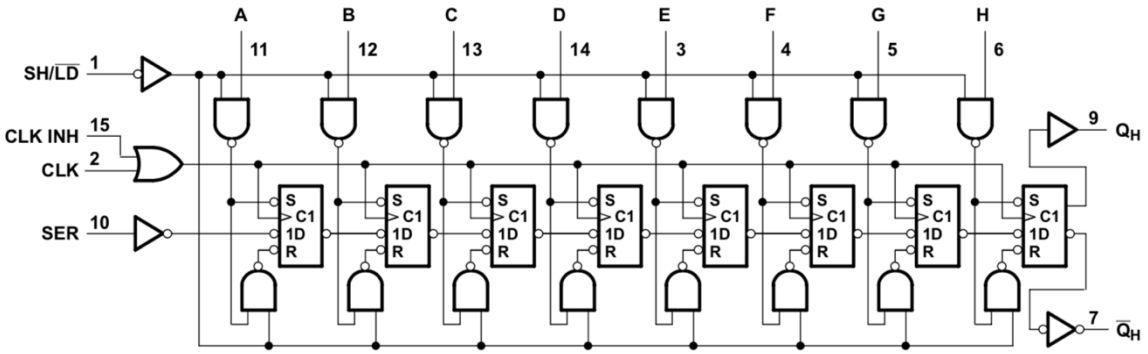


Figure 8: Logic diagram of SN74HC165 Parallel-in/Shift-out [23]

According to the datasheet provided by Texas Instruments for this device, as shown in fig-8 parallel-in access to each stage is provided by eight individual direct data (A–H) inputs that are enabled by a low level at the shift/load (SH/LD) input. Clocking is accomplished by a low-to-high transition of the clock (CLK) input while SH/LD is held high and CLK INH is held low. The functions of CLK and CLK INH are interchangeable. Since a low CLK and a low- to-high transition of CLK INH also accomplish clocking, CLK INH should be changed to the high level only while CLK is high. Parallel loading is inhibited when SH/LD is held high. While SH/LD is low, the parallel inputs to the register are enabled independently of the levels of the CLK, CLK INH, or serial (SER) inputs [23] [15].

This device was operated with driver software to automate parallel to serial conversion. The HC165 shift register IC was connected to a Raspberry Pi, and it was operated using a python driver to collect digital data from the PLC. The timing diagram for SN74HC165 shown in fig-9 was referred to develop the logic of python driver for HC165. The timing diagram illustrates the logical sequence and the timing sequence of the control signals of the HC165 shift registers that need to be set to logical HIGH or logical LOW in order to achieve the parallel to serial conversion.

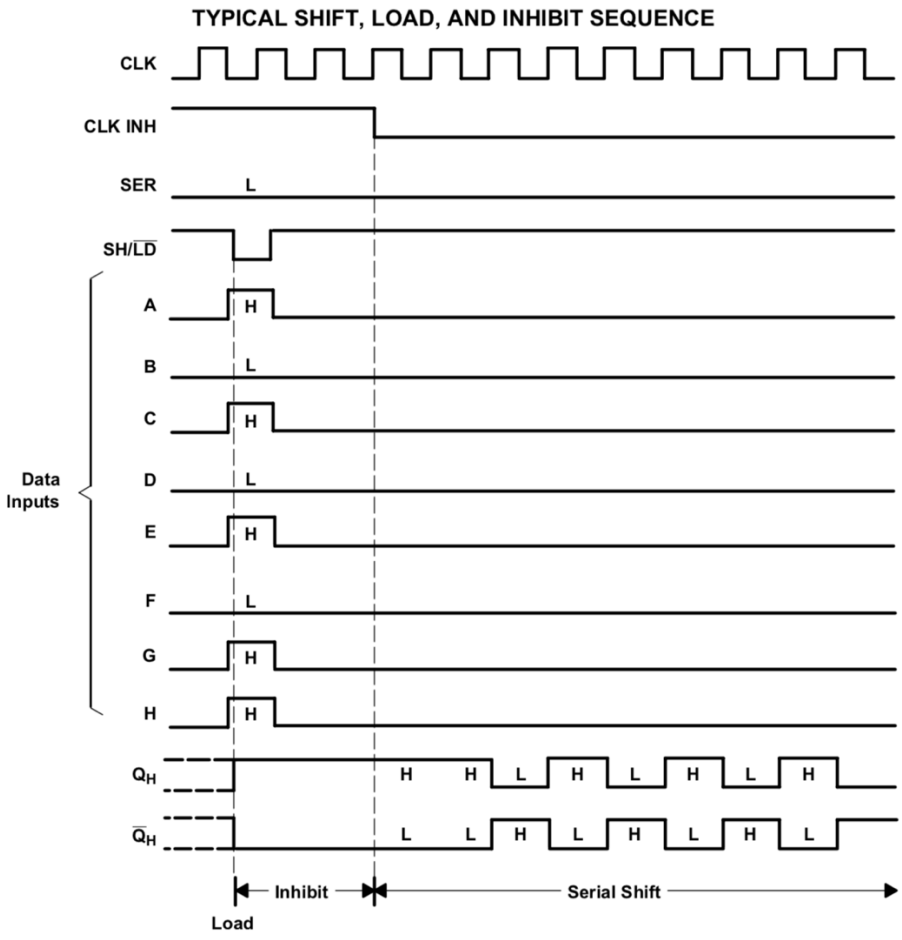


Figure 9: Timing Diagram of SN74HC165 [23]

The goal of the driver program is to automate and synchronize control signals CLK, CLK INH, SER, and SH/LD to achieve parallel to serial conversion. The shift register IC is made of clocked SR latches, as shown in fig-8. These latches are made of two cross-coupled NOR gates or two cross-coupled NAND gates [15]. Upon receiving a clock pulse, the input to these latches is transferred to the output. As shown in fig-9, Inputs to the shift registers are parallel inputs (A from H) that need to be converted into the serial output. The CLK INH signal disables the CLK signal when it is set to HIGH. While CLK INH is set to HIGH (also called the inhibit period for shift registers), SH/LD needs to be set to LOW to load inputs at A to H into SR latches. Once the values are loaded into SR latches, SH/LD needs to be set HIGH to prepare SR latches for the shift

operation. Now CLK INH could be set to LOW again to start the serial shift. The shift registers will start shifting the values into SR latches towards QH, one bit per clock cycle. The values on QH can be collected as the serial on every clock cycle. SER is needed to be LOW to enable the IC. This control signal can be used to synchronize the shifting operation between multiple shift registers. We followed the sequence explained above to develop a python code to drive the shift registers to convert parallel data into serial data.

Our driver program is posted in Appendix-A. This program simulates the timing diagram to perform the parallel data to serial data conversion using Raspberry Pi's GPIO port with proper connections made with the shift registers to automate the control signals. The "RPi.GPIO" module was a convenient tool for python to use Raspberry Pi's GPIO port. It provided functions such as *GPIO.RPi.setup()*, *GPIO.RPi.output()*, and *GPIO.RPi.input()* to operate on the GPIO port. Pins A from H on the shift registers were connected to PLC's digital I/O ports through the interface card. QH and other control signals were connected to the Raspberry Pi's GPIO port. The driver program simulated the timing diagram (fig-9), converted parallel data from the PLC's digital I/O ports into the serial data, transformed the captured data into the format that was compatible with training algorithms, and stored it into files named input.txt, output.txt, and all.txt. These files were used as the training data for the NN core. In fig-6, there is a scanning rate block connected to shift registers, which is the capture time interval explained earlier in this chapter. In the experiment, the scanning rate for the shift registers was interpreted as how often the control signal SH/LD was used to load the parallel inputs into the shift registers and the time period between two load operations was considered as the capture time interval. Our program wrote a new row of data in the training data file at the end of every capture time interval.

3.3.2 HARVESTING ANALOG VALUES:

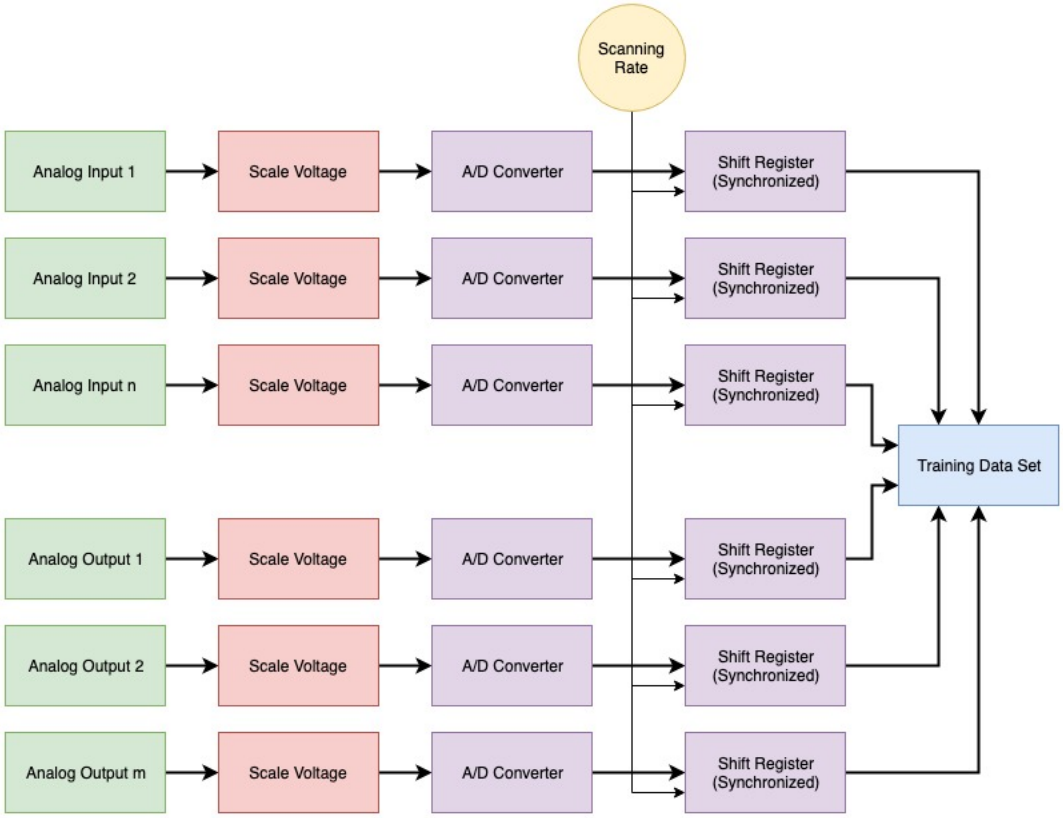


Figure 10: Harvesting analog values

Harvesting analog values for the training data set proved to be similar to harvesting digital values. The only difference was that it required a dedicated 8-bit shift registers IC for each analog input paired with analog to digital converter (A/D converter). As shown in fig-10, voltage scaling was required to get the analog voltage in the range of supported voltage rating for shift registers. Raspberry Pi and Arduino are digital devices; hence, analog values must be converted into digital form so that these devices can process them. The combination of analog to digital converter and shift registers were used to convert analog values into digital values.

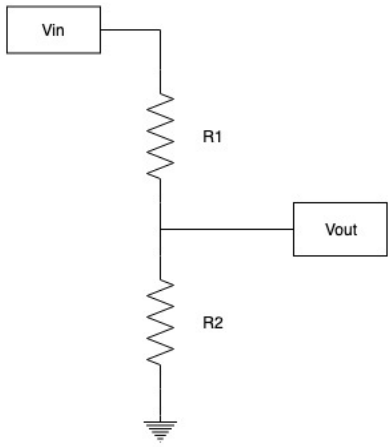


Figure 11 Voltage divider

To linearly scale the voltage, a simple circuit called "voltage divider" was used. It is a circuit which has linear characteristics, and it scales down voltage with a constant which can be calculated using the following equation [14].

$$Vout = (\frac{R2}{R1 + R2})Vin$$

Usually, a PLC's analog modules operate in range of 0 V from 24 V, and A/D converters operate in range of 0 V from 5 V. The voltage scaler was designed keeping Vin as 24 V and Vout as 5 V. The value of resistors R1 and R2 were set in such a way that satisfied the voltage divider equation.

The next step was to Implement A/D converter and shift registers. MCP3004/3008 devices developed by Microchip Technology Inc., are successive approximation 10-bit Analog to Digital (A/D) converters with on-board sample and hold circuitry [17]. They come with built-in shift registers to output digitally converted analog data. This chip was used with Raspberry Pi and run using a python program to retrieve analog data. The operation is very similar to shift registers 74HC165, and the driver code for MCP3008 can be integrated into the driver program to harvest

digital I/O. Adafruit® has developed a python library to use MCP3008 with Raspberry Pi called Adafruit_MCP3008, which was used to collect an analog value In the experiment [31].

Since the PLC that was used in the experiment did not have the analog module, we tested the capturing of an analog signal with an external variable power supply and checked if it converted analog values into digital format to verify that the obtained value could be stored in the training data set. The analog values were not part of the neural network training. The purpose of this section is to introduce a method to collect analog values in case if the PLC had an analog module.

3.3.3 HARVESTING TIMER VALUES:

Different PLC manufacturers have designed software to program their own PLCs. The software provides an interface to monitor values of timers and variables during the execution of programs on PLC, which means they read PLC's core to obtain these values. If manufacturers decide to implement neural network integration to their PLC, they will have to modify their monitoring software to store timer values into the training data set. We used a Click PLC and the Click programming software in the lab setup to perform the experiment. Code for this software is highly protected. Therefore, we were not able to modify the code to obtain timer values. Hence, software integration to the programming software is necessary to get timer values. In the experiment, timer values were not stored in the training data set.

3.4 CHALLENGE#2:

Problem description: *Identify the correct neural network architecture and training algorithm that will predict outputs from inputs.*

A neural network is a complex network of more than one neuron. In machine learning terminology, a neuron is considered as the fundamental element that represents the characteristics of a neural network. In more straightforward terms, a neuron represents a mathematical function with one or more inputs and an output. These neurons can have connections with other neurons. The neural network architecture is the organization of these neurons, and it represents how each neuron is connected with other neurons. Training algorithms are mathematical approaches to calculate offset-weights on each input to a neuron using error corrections through iterative methods. The efficiency of training algorithms depends on the architecture of neurons and the type of neurons implemented within the network.

Neural networks are mighty as nonlinear signal processors, but obtained results are often far from satisfactory [28], meaning that there are a few challenges related to neural network's architecture and training algorithms that need to be addressed to make neural network applications successful. The following are the critical challenges to neural network applications [28]:

1) Which neural network architecture should be used?
2) How vast should a neural network be?
3) Which learning algorithms are most suitable for a chosen architecture?

Addressing these challenges achieved the successful training of the neural network.

For our NNPLC application, the neural network architecture can be designed from the understanding of ladder logic implemented in the PLC. This approach was only useful for smaller PLC applications, but while dealing with complex ladder logic, implementation of standardized neural network architectures was crucial for achieving successful training. When manually derived architectures became too complex, the available training algorithms were not able to

successfully train the neural network. Below is an example of a neural network derived from the ladder logic [18].

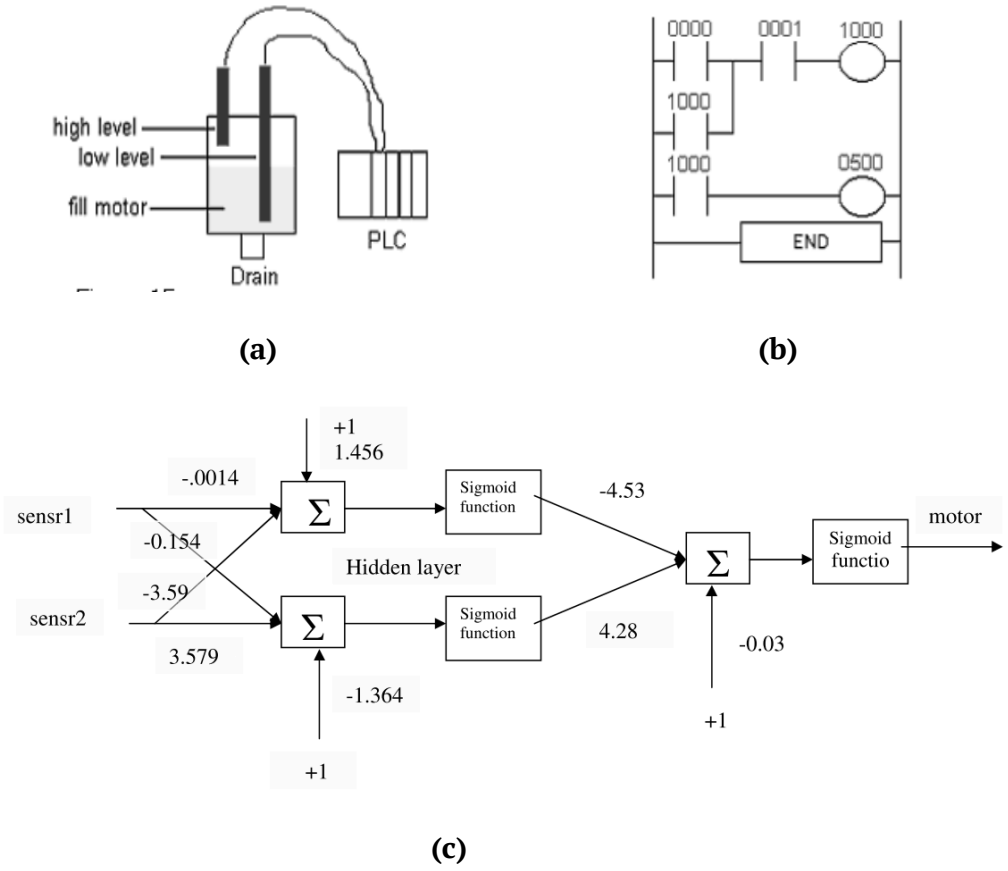
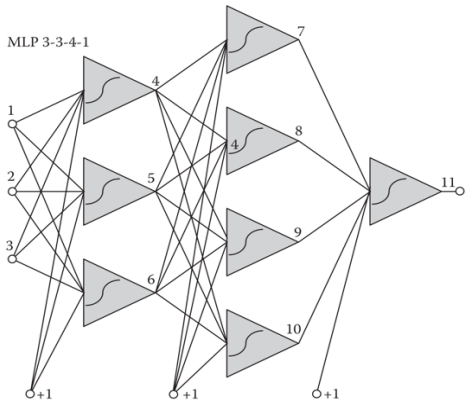


Figure 12: Neural Networks architecture for automated oil pump [18]

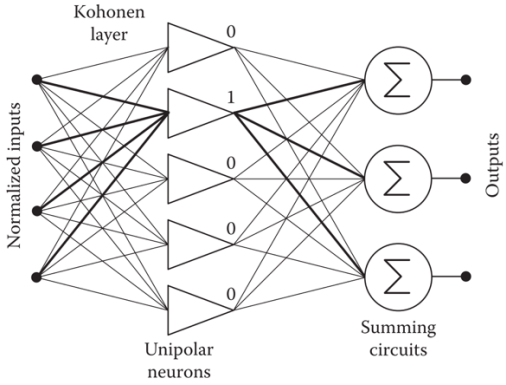
Fig-12(a) shows the configuration of an automated oil pump. It has two sensors to measure the amount of oil in the tank; one to indicate a high level of oil and the other to detect low level of oil. These indicators/sensors are inverted binary logic switches with HIGH and LOW digital states. The motorized pump starts to fill up the tank as soon as the low-level indicator's state turns to digital LOW. The motor is turned off when the oil reaches to the high-level indicator, and its state turns to digital HIGH. This logic is implemented in the ladder logic, as shown in fig-12(b). Ladder variable 0000 is the low-level indicator, and 0001 is the high-level indicator. Variable 1000 is a logic variable which controls the push button on the motor via PLC's

physical port accessed by the variable 0500. The motor is turned off when high-level and low-level indicators are on state HIGH and HIGH, respectively. The motor is turned on when both indicators are on state LOW. Motor's operation is not interrupted when the low-level indicator is HIGH, and the high-level indicator is LOW. This logic is very similar to the parity-2 problem for neural networks [26]. As shown in fig-12(c), the neural network architecture to solve the parity-2 problem is implemented and trained to duplicate the logic written in ladder logic. Weights and biases within the neural network architecture are determined after training [18]. Explained above was an example of simple ladder logic. This approach might get too complicated when ladder logics are complex.

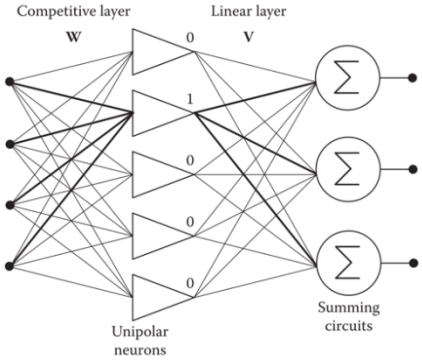
There are neural network architectures already developed for optimal training of complex functions. These architectures are derived through researches done over a long period of time on deep learning and artificial intelligence.



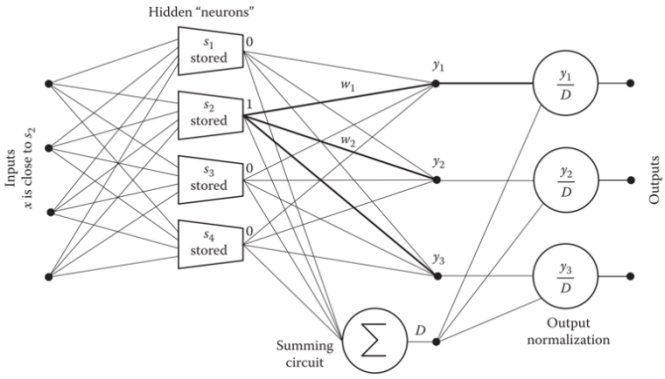
(a) Multi-layer perceptron network (MLP)



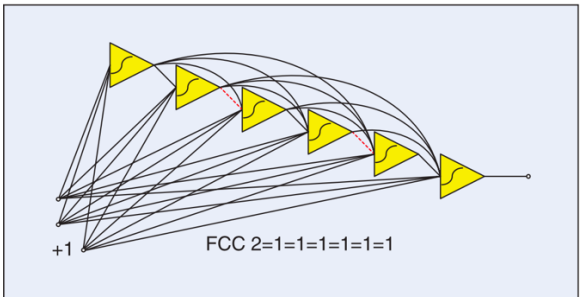
(b) Counterpropagation network



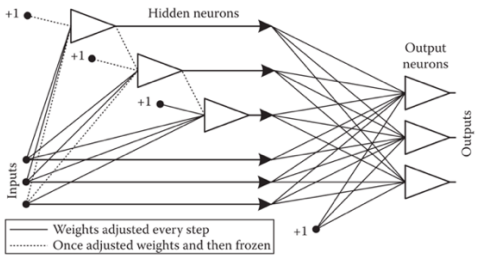
(c) Linear Vector Quantization



(d) Radial basis functions network (RBF)



(e) Fully Connected Cascade (FCC)



(f) Cascade correlation architecture

Figure 13: Standard Neural Network Architectures [27] [28]

Fig-13 shows neural network architectures that are popular among researchers. The multi-layer perceptron (MLP) architecture, also known as a simple feedforward network, is the preferred neural network topology by researchers [1]. It is the oldest neural network architecture, and it is compatible with many training software [28]. This architecture usually gives satisfying results, but in the case of a large number of neurons, it could take a significant amount of time to train. There are several neural network architectures like counterpropagation (fig-13(b)), RBF(fig-13(d)), linear vector quantization(fig-13(c)), Fully connected cascade (fig-13(e)), cascade correlation architecture(fig-13(f)) that can be used for rapid prototyping, but they require large number of neurons. In most cases, these architectures require signal-normalization processes [28]. The performance of any neural network architecture depends on the type and the amount

of data put in the training data set. For example, if the training dataset contains a small number of patterns, then the counterpropagation architecture gives optimal performance because with unipolar activation function in the first layer, the network works as a look-up table. When the linear activation function (or no activation function at all) is used in the second layer, then the network also can be considered as an analog memory [27].

How to choose a neural network architectures for our NNPLC application? The content of the training data set is one of the essential factors to narrow down to a few suitable neural network architectures for the ladder logic implemented in the PLC. There are a few architectures which are efficient to train the digital data, and there are a few of them better suited to train non-digital data. The traditional approach to pick an architecture was used after the search was narrowed down to a few suitable architectures. In this conventional approach, neural network topologies/architectures are, in most cases, selected by a trial-and-error process [28]. Often, success depends on a lucky guess; hence, the search process started with larger architecture, and the network was pruned in a more or less organized way [4]. Since there was only the digital data to train in the experiment, the search was narrowed down to a few architectures among which the counterpropagation and the simple feedforward architectures showed the best results. We chose to implement a simple feedforward architecture because counterpropagation architecture's performance degrades for more extensive training data set [27]. The feedforward architecture is also known to work well with the Levenberg-Marquardt algorithm which is the fastest training algorithm developed till this date.

The next step was to choose a training algorithm to train the chosen architecture. The procedure to carry out the learning process in a neural network is called the optimization/training algorithm [29]. There are many algorithms available to train different

architectures of neural network. They vary in terms of memory requirements, speed, and precision. The training of the neural network is achieved through the minimization of a loss index; which is a function that measures the performance of a neural network on the training data set. The loss index is made of an error term and a regularization term. The error term evaluates how neural network fits the data set, and the regularization term prevents overfitting by controlling the effective complexity of the neural network [29]. The goal of training algorithms is to minimize the loss index by applying various error corrections. Prevention of overfitting is often done with the combination of pruning algorithms and training algorithms. They prune the architecture in terms of several neurons and the number of connections within the neural network.

As mentioned earlier there are many training algorithms available and some of the popular algorithms are Error Back Propagation algorithm (EBP) [20], Levenberg-Marquardt algorithm (LM) [16], Neuron-By-Neuron algorithm (NBN) [25], Bayesian Regularization algorithm [24][11] and Scaled Conjugate Gradient [5]. We theoretically compared these algorithms to determine what algorithms to choose to test the boundries of the training efficiency of NNPLC application.

***3.4.1 ERROR BACKPROPAGATION ALGORITHM (EBP)*:** [20]

Backpropagation method of training neural networks is widely accepted by researchers. It is the most used algorithm to train complex neural network architectures. The procedure repeatedly adjusts the weights of connections in the network to minimize a measure of the difference between the actual output vector of the net and the desired output vector [20]. This method of training is particularly useful when there are hidden layers introduced in the neural network architecture. As a result of the weight adjustments, internal 'hidden' units which are not

part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure.

In this method, during the first iteration of training, weights on each connection within neurons are randomly generated. Total error in the performance of the network with a particular set of weights can be computed by comparing the actual and desired output vectors for every case or training pattern. The total error E is defined as,

$$E = \frac{1}{2}\sum_{c}\sum_{j}\left(y_{j,c} - d_{j,c}\right)^2$$

where c is an index over cases (input-output pairs or training pattern), j is an index over output units, y is the actual state if an output unit and d is its desired state [20]. In the traditional approach to error back propagation, total E is targeted to be minimized by gradient decent where partial derivative to E with respect to each weight in the network is computed. This is simply the sum of the partial derivatives for each training patterns or input-output cases. For a given case partial derivatives are calculated in two passes. For the forward pass, the state of units (or neurons) in each layer are determined by the input they receive from lower layers using the following equations,

$$x_j = \sum_i y_i w_{i,j}$$

$$y_j = \frac{1}{1 + e^{-x}}$$

Where The total input, x_i, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights w_{ji} on these connections. A unit has a real-valued output, y_j,

which is a non-linear activation function of its total input. By applying partial derivation with respect to total inputs, outputs, and individual weights and taking into account all the connections emanating from the unit I, we have the backward pass which propagates derivatives from the top layer back to the bottom one,

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j \cdot w_{j,i}}$$

We have now seen how to compute $\frac{\partial E}{\partial y}$ for any unit in the penultimate layer when given $\frac{\partial E}{\partial y}$ for all units in the last layer. We can, therefore, repeat this procedure to compute this term for successively earlier layers, computing $\frac{\partial E}{\partial w}$ for the weights as we go.

One way of using $\frac{\partial E}{\partial w}$ is to change the weights after every input-output case. This has the advantage that no separate memory is required for the derivatives. An alternative scheme, which we used in the research reported here, is to accumulate $\frac{\partial E}{\partial w}$ overall the input-output cases before changing the weights. The simplest version of gradient descent is to change each weight by an amount proportional to the accumulated $\frac{\partial E}{\partial w}$.

$$\Delta w = -\varepsilon \frac{\partial E}{\partial w}$$

Here, ε is the proportional relation constant and is called the learning rate of the backpropagation.

By applying this weight change Δw in every iteration, the total error E is minimized, and the network is trained for optimal performance. This is the traditional approach to error back propagation, and it is also known as the steepest-decent gradient algorithm.

3.4.2 LEVENBERG-MARQUARDT ALGORITHM (LM): [16][9][29]

Levenberg-Marquardt algorithm combined with back-propagation is the fastest training algorithm till this date. It is an improved version of Newton's method to minimize the loss function of the neural network. Newton's method is nothing but the second order partial derivation method to find the minima of a function. Since there are too many weights in the network and each one needs to be corrected, it is required to have a partial derivation of errors with respect to each weight. Hence, Newton's method of training uses a Hessian matrix, which is a matrix of all possible partial derivatives. This is a primitive way to train the network and often gives non-satisfactory results since the learning rate as the error decreases are not factored in. Also, it requires too many precise calculations. The Levenberg-Marquardt algorithm is an approximation to Newton's method.

This algorithm uses the squared function of errors just as discussed in the error backpropagation algorithm. Donald W. Marquardt said, "Most algorithms for the least-squares estimation of non-linear parameters have centered about either of two approaches. On the one hand, the model may be expanded as a Taylor series and corrections to the several parameters calculated at each iteration on the assumption of local linearity. On the other hand, various modifications to the method of steepest-descent have been used. Both methods not infrequently run aground, the Taylor series method because of divergence of the successive iterates, the steepest-descent (or gradient) methods because of agonizingly slow convergence after the first few iterations" [16].

The algorithm follows the same procedure as the error backpropagation. Forward computation is done by applying inputs to the network, and the output to each neuron is calculated using connected weights to the neuron and its activation function. For backward

computation; the loss function f of the neural network for this algorithm is defined as the sum of squared errors e,

$$f = \sum_{i=0}^{a} e_i^2$$

Where a is the number of training patterns/input-output pairs in the training data set.

This is the approximation of Newton's method; hence, it works without computing the exact Hessian matrix by approximating the Hessian matrix using the gradient vector and the Jacobian matrix. In mathematical terms, the Jacobian matrix is the matrix of all first-order partial derivatives of a vectored valued function. The Jacobian matrix of the loss function with respect to weights can be expressed as,

$$J_{i,j} = \frac{\partial e_i}{\partial w_j}$$

Here, i goes from 1 to a and j goes from 1 to b, where a is the number of training patterns in the data set, and b is the number of weights in the neural network architecture.

The gradient vector of the loss function is calculated as,

$$\nabla f = 2J^T \cdot e$$

Hence, the Hessian matrix is approximated as,

$$Hf \approx 2J^T \cdot J + \lambda I$$

Where, λ is a damping factor that ensures the positiveness of the Hessian and I is the identity matrix. Evaluating this approximated Hessian matrix yields factors to modify weights which are simplified as [16][9],

$$w^{(i+1)} = w^{(i)} - \left(J^{(i)T} \cdot J^{(i)} + \lambda^{(i)} I\right)^{-1} \cdot \left(2J^{(i)T} \cdot e^{(i)}\right), \qquad i = 0,1,2, \ldots$$

Here, i represents the index of connected weights.

When the damping parameter λ is zero, this is just Newton's method, using the approximate Hessian matrix. On the other hand, when λ is large, this becomes gradient descent with a small training rate [29].

The parameter λ is initialized to be large so that first updates are small steps in the gradient descent direction. If any iteration happens to result in a failure, then λ is increased by some factor. Otherwise, as the loss decreases, λ is decreased, so that the Levenberg-Marquardt algorithm approaches the Newton method. This process typically accelerates the convergence to the minimum [29].

Despite being the fastest algorithm, it has one major drawback. The Jacobian matrix can get too complex for larger data sets, and therefore, it requires a large amount of memory during the execution of training. Hence, it is not ideal for larger training data sets and complex neural network architectures in some cases.

3.4.3 NEURON-BY-NEURON ALGORITHM (NBN): [25]

It is a similar algorithm to Levenberg-Marquardt modification to error backpropagation. This algorithm introduces a new NBN method for calculating the gradients and Jacobian matrix for arbitrarily connected feedforward networks. The rest of the computations for weight updates and optimization remains the same as the LM algorithm.

In the forward calculation, the neurons connected to the network inputs are first processed so that their outputs can be used as inputs to the subsequent neurons. The follow- ing neurons are then processed as their input values become available. In other words, the selected computing sequence has to follow the concept of feedforward networks and signal propagation. If a signal reaches the inputs of several neurons at the same time, then these neurons can be processed in any sequence.

Backward computation starts with the last neuron and continues toward the input. Also, for this algorithm, the concept of attenuation vector (a) is introduced. The attenuation vector (a) represents signal attenuation from a network output to the outputs of all other neurons. The size of this vector is equal to the number of neurons. Backward computation process starts with the values of one assigned to the last element of the (a) vector and zeroes to the remaining output neurons. During backward processing for each neuron, the value of the delta of this neuron is multiplied by the slope of the neuron activation function and then multiplied by neuron input weights. The results are added to the other elements of the (a) vector neurons which are not yet processed.

The size of the (a) vector is equal to the number of neurons, while the number of Jacobian elements in one row is much larger and is equal to the number of weights in the network. To obtain all row elements of the Jacobian for the pth pattern and oth output, a very simple formula can be used to obtain the element of the Jacobian matrix associated with the input k of neuron n,

$$\frac{\partial e_{p,o}}{\partial w_{n,k}} = d(n)_{p,o} \cdot s(n)_p \cdot node(k)_{p,o}$$

Where d(n) is the element of the (a) vector, and $s(n)$ is the slope calculated during aaforwarding computation, with both of them being associated with neuron n. The $node(k)$ is the value on the kth input of this neuron.

The process is repeated for every pattern, and if a neural network has several outputs, it is also repeated for every output. The process of gradient computation is the same as the LM, but instead of storing values in the Jacobian matrix, they are being summed into one element of the gradient vector,

$$g(n,k) = \sum_{p=1}^{P} \sum_{o=1}^{O} \frac{\partial e_{p,o}}{\partial w_{n,k}} \cdot e_{p,o}$$

Since the Jacobian is not stored into the memory and used in the computation of the gradient vector as we go, it moderately reduces the memory requirements compared to LM. Also, since it is not stored in the memory, repeated calculation of the same element can occur, which reduces the performance.

***3.4.4 SCALED CONJUGATE GRADIENT*:** [5][29]

The conjugate gradient method can be regarded as something intermediate between gradient descent and Newton's method. It is motivated by the desire to accelerate the typically slow convergence associated with gradient descent. This method also avoids the information requirements associated with the evaluation, storage, and inversion of the Hessian matrix, as required by Newton's method.

Since learning in realistic neural network applications often involves adjustment of several thousand weights, only optimization methods that apply to large-scale problems are relevant as alternative learning algorithms. The general opinion in the numerical analysis community is that especially one class of optimization methods, called the Conjugate Gradient Methods, are well suited to handle large-scale problems effectively [3].

Let's start with the basic notations for the algorithm to understand to strategy to implement this algorithm.

Let an arbitrary feedforward neural network be given. The weights in the network will be expressed in vector notation. A weight vector is a vector in the real Euclidean space $\mathcal{R}^N$, where N is the number of weights and biases in the network. A weight vector will often be referred to as a point in $\mathcal{R}^N$ or just a point in weight space. Let $\vec{w}$ be the weight vector.

$$\vec{w} = \overrightarrow{w_1} + \alpha_1\overrightarrow{p_1} + \cdots + \alpha_k\overrightarrow{p_k}$$

Where, $\overrightarrow{w_1}$ is a point in weight space. Notations $\overrightarrow{p_1}, \overrightarrow{p_2} \ldots, \overrightarrow{p_k}$ is a subset of a conjugate system which is a set of nonzero weigh vectors in $\mathcal{R}^N$.

We assume that a global error function $E(\vec{w})$ depending on all the weights and biases is attached to the neural network. $E(\vec{w})$ could be the standard least square function or any other appropriate error function. $E(\vec{w})$ can be calculated with one forward pass and the gradient $E'(\vec{w})$ with one forward and one backward pass. $E'(\vec{w})$ is defined as,

$$E'(\vec{w}) = \left(\ldots, \sum_{p=1}^{P} \frac{dE_p}{dw_{i,j}^{(l)}}, \sum_{p=1}^{P} \frac{dE_p}{dw_{i+1,j}^{(l)}}, \ldots, \sum_{p=1}^{P} \frac{dE_p}{dw_{N_1,j}^{(l)}}, \sum_{p=1}^{P} \frac{dE_p}{d\theta_j^{(l+1)}}, \sum_{p=1}^{P} \frac{dE_p}{dw_{i,j+1}^{(l)}}, \ldots \right)$$

Where, $w_{i,j}^{(l)}$ is the weight from unit number i in layer number 1 to unit number j in layer number $l+1$, N_1 is the number of units in layer j, and $\theta_j^{(l+1)}$ is the bias for unit number j in layer number $l+1$. P is the number of patterns presented to the network during the training and E_p is the error associated with pattern p.

The minimization is a local iterative process in which an approximation to the function in a neighborhood of the current point in weight space is minimized. The approximation is often given by a first or second order Taylor expansion of the function. The following steps are executed to minimize the global error function $E(\vec{w})$.

1. Choose initial weight vector $\overrightarrow{w_1}$ and set $k = 1$.
2. Determine a search direction $\overrightarrow{p_k}$ and a step size α_k so that $E(\overrightarrow{w_k} + \alpha_k\overrightarrow{p_k}) < E(\overrightarrow{w_k})$
3. Update vector: $\overrightarrow{w_{k+1}} = \overrightarrow{w_k} + \alpha_k\overrightarrow{p_k}$.
4. If $E'(\overrightarrow{w_k}) \neq \hat{0}$ then set $k = k + 1$ and go to step 2 else return $\overrightarrow{w_{k+1}}$ as the desired minimum.

Determining the next current point in this iterative process involves two independent steps. First, a search direction has to be determined, i.e., in what direction in weight space do we want to go in the search for a new current point. Once the search direction has been found, we have to decide how far to go in the specified search direction, i.e., a step size has to be determined.

To put this mathematics in words, the data set is randomized and divided into subcategories and multiple training iterations are applied for each subcategory. As we move forward from each category the scaling factors for the conjugates to adjust the weights is optimized to reach to the minima of the global error function $E(\vec{w})$ faster when compared to the steepest-descent gradient method for error backpropagation. This is also referred to as the modified version of error backpropagation through gradient descent for fast supervised learning.

3.4.5 CHOOSING TRAINING ALGORITHMS FOR NNPLC:

After understanding the mathematics, advantages, and disadvantages of training algorithms explained above, it is concluded that the LM algorithm is the fastest algorithm but it used the most amount of memory, whereas the EBP with gradient descent is the slowest of all, but it requires the least amount of memory since the Hessian and the Jacobian matrices are involved in computation. The research shows that the scaled conjugate method of training network requires the same amount of memory, but it is faster than the steepest-gradient decent error-backpropagation algorithm. The traditional EBP training process requires 100 to 1,000 times more iterations than more advanced algorithms such as Levenberg–Marquardt (LM) or neuron by neuron (NBN) [28]. To test the NNPLC application of neural network, EBP with a scaled conjugate gradient, which was the slowest, but it had the least memory requirements, and the LM algorithm, which was the fastest, but it had larger memory requirements, were chosen to

test the boundaries of training efficiency. Both of these algorithms worked well with the feedforward architecture of the neural network, and they both yielded accurate results.

3.5 CHALLENGE#3:

Problem description: Employing the prediction model to discriminate normal from abnormal PLC operations.

This challenge was the last critical challenge to successfully implement the neural network with PLC operations. There is often a leftover training error, which is very small in value, after the training process has successfully completed. If the training process has achieved a small acceptable error, then wrong predictions would occur rarely, but the possibility of such occurances would still exist. Therefore while comparing the outputs generated by the PLC with the outputs generated by the neural network, distinguishing between the error caused by abnormal PLC operation and the error caused by the wrong prediction from the neural network was a critical challenge to be addressed.

Determining how the trained neural network could be deployed in parallel with the operating PLC helped address this challenge. We categorized the deployment into two categories.

1. Dynamically deployed
2. Statically deployed

These methods are different ways to execute the trained neural network and compare the outputs generated from the PLC and the NN core.

3.5.1 DYNAMICALLY DEPLOYED:

In this method of deployment, the neural network runs in parallel with the PLC in real time. The operating PLC and the trained neural network, both get the same inputs to generate output at the same instant. Outputs produced by both the PLC and the neural network core are dynamically compared for each input pattern. The difference in output, if any, is recorded in real time. Usually, malfunction errors of edge devices and errors caused by abnormal execution of PLC's code would differ significantly from the data used to train the neural network. Thus, causing significantly large errors compared to errors caused by the neural network due to the training error. Although, there may be a few cases where it is difficult to determine what caused the error.

To solve the problem, the concept of averaging the root mean squared value of errors is used. Root mean squared value of the error for each input pattern is continuously averaged, recorded, and plotted. If the plot continues to rise above the training error, then it is concluded that there is a malfunction in the PLC operation since errors caused by wrong predictions from the neural network would keep converging back to the almost value of training error.

3.5.2 STATICALLY DEPLOYED:

In this method of deployment, the trained neural network stays idle while the values of inputs and outputs from the PLC are recorded for a predefined interval of time. At the end of the time interval, a driver feeds the recorded data to the trained neural network. The neural network generates outputs for each input pattern. The testing error is calculated by averaging the root mean squared value of the differences from recorded actual PLC outputs and neural network outputs. This process is repeated for every predefined interval while the PLC is in operation. The testing error is continuously averaged, recorded, plotted, and observed. Before deploying the

neural network with the real system, testing errors from a set of randomly chosen patterns from the training data set is calculated. This process is repeated for multiple sets of different sizes. These random sets must be chosen in such a way that the average testing error for each set is more than the training error. The final testing error is calculated by averaging the testing errors from all sets. Suppose the predefined testing error, E_p, and the error calculated at the end of every predefined time interval actual testing error, E_a. The training error obtained during the training process is defined as E_T. The prediction model to discriminate normal from abnormal PLC operations can be derived as:

$$E_a \leq E_T \rightarrow Normal\ operation$$

$$E_T < E_a < E_p \rightarrow High\ probability\ of errors\ from\ neural\ network$$

$$E_a \geq E_p \rightarrow Most\ likely\ abnormal\ operation$$

The final decision is made by observing E_a. The value of E_a staying above E_T, implicates that averaging of the actual training error is not converging back to the training error found during the training process. Meaning, the error was the result of a malfunction in the system or an abnormal PLC operation.

3.5.3 COMPARISON:

Both methods have their advantages and disadvantages. Dynamically deployed method continuously calculates errors in real time. Hence, there is no delay in the detection of abnormal operation. This method requires to have synchronization between the neural network and the PLC to make sure that outputs from both the PLC and the NN core are generated and compared at the same time. Hence, it requires more efforts to implement. The statically deployed method does not require any synchronization between the PLC and the neural network. But there could be a delay in the detection of abnormal operation since the new error values are evaluated after

a time interval. The static deployment factors-in the predefined testing error E_p. As described above, it can be used to predict warnings to alert operators of the automation system about the possibility of abnormal operation. We tested the NNPLC application using the static deployment method since it was easier to implement, and it had more parameters to validate the work. The statically deployed neural network seemed more practical approach to validate the research.

3.6 TESTING PLATFORMS FOR NEURAL NETWORKS:

There are many tools available to implement and test neural networks applications such as MATLAB, NeuroSolutions infinity, NBN trainer, etc. There are programming libraries such as NumPy, TensorFlow, Blocks, NeuPy, etc., for the python programming language to implement neural networks. These libraries contain functions to operate on Hessian and Jacobian matrices, to perform vector algebra, to generate architecture matrices, and henceforth.

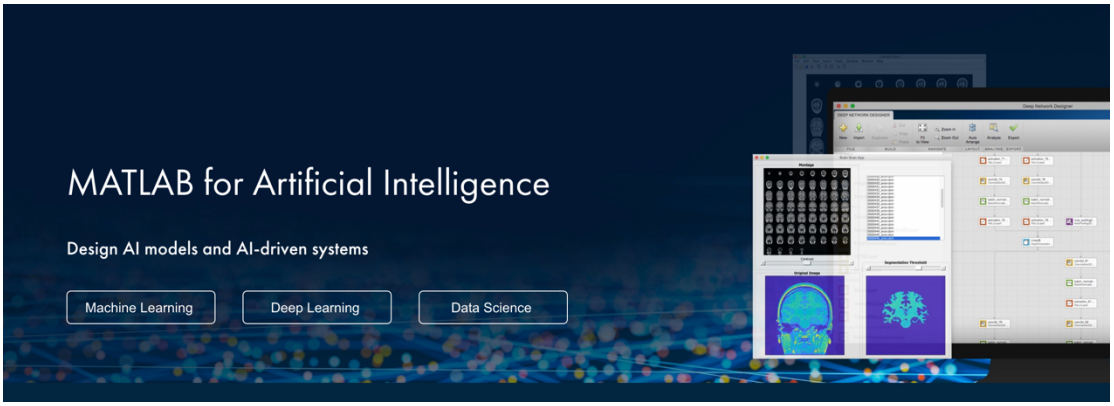


Figure 14: MATLAB [32]

In the experiment, MATLAB's neural network fitting tool (nftool) was used to develop, implement, and test the neural network. This tool supported the implementation of simple feedforward architecture with hidden layers and with the customizable number of neurons in the architecture. It also provided options to train the network using scaled conjugate gradient and

LM algorithms. It offered a benefit to test the network with random test cases from the training data set, which was useful in implementing the statically deployed prediction model.

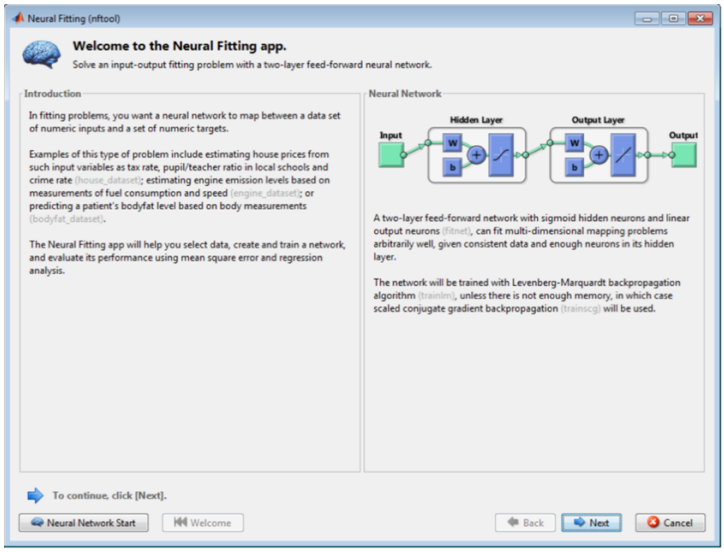


Figure 15: Neural network fitting tool (nftool) [32]

The nftool offered GUI to design, configure, train, and test the network, which made it easier to use and demonstrate the results. It calculated numerous parameters such as the training error, the testing error, error reduction plots, convergence plot, etc., which proved to be advantageous in evaluating the performance of the network. Such features made this tool ideal for the research.

4. CHAPTER-4: SOLUTION VALIDATION

This chapter shows the validation results for the solution described in chapter-3.

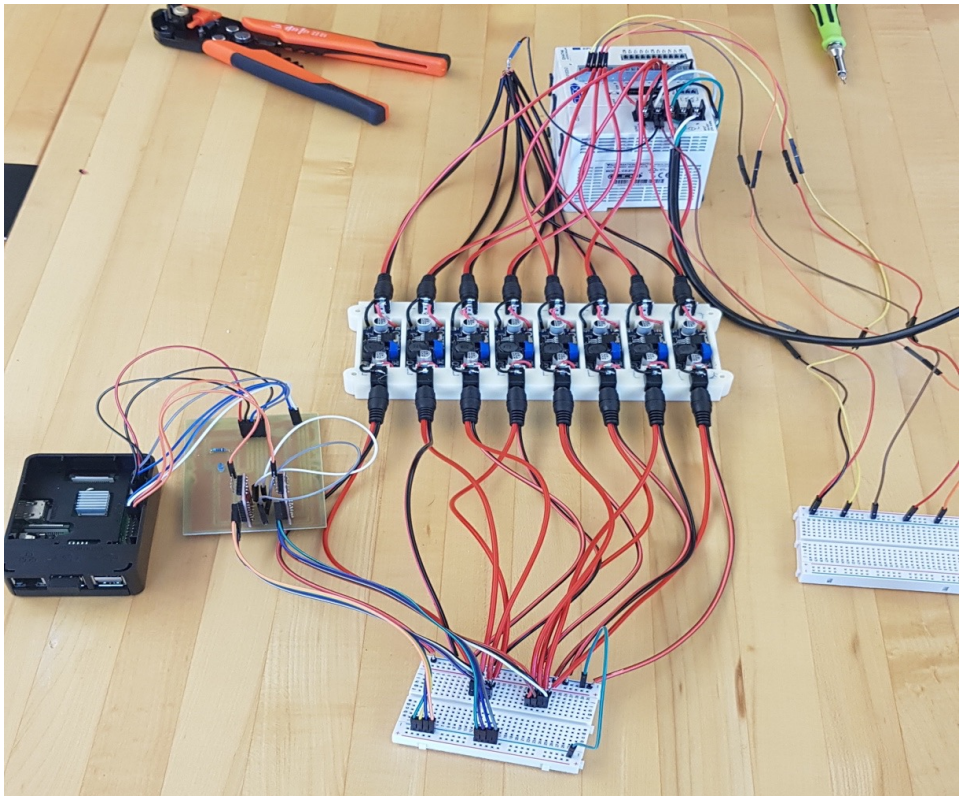


Figure 16: Lab setup of NNPLC

Fig-16 is an actual picture of the setup we built to validate the solution.

4.1 APPARATUS AND SOFTWARE USED FOR NNPLC:

Below is the list of hardware that was used to implement the NNPLC for validation.

1. Click PLC (model# C0-01DD2-D)

2. Raspberry Pi 3 model B+
3. Texas Instruments® 74HC165 shift registers
4. E-boot LM2596 voltage step-down converters

These are the necessary hardware required to implement the NNPLC. Some additional hardware to build electrical circuits such as printed circuit boards, jumper wires, breadboards, etc., are not mentioned above.

Below is the list of software used for the validation.

1. MATLAB R2017a
2. Click Programming Software (Version 2.40)
3. Python 2.7
4. Microsoft Excel 2016

4.2 VALIDATION OF THE CONCEPT:

This section contains validation results for the three critical problems explained in chapter-3. The results were obtained during the experiment performed using the Click PLC C0-01DD2-D, that belongs to a family of Click PLCs with digital I/O ports. The entire validation was done from the digital data obtained. Analog values were not used in the ladder logic used for the validation. Timer counts were also not harvested and stored in the training data set since certain limitation to access the programming software were encountered.

4.2.1 LADDER LOGIC IMPLEMENTED IN THE PLC:

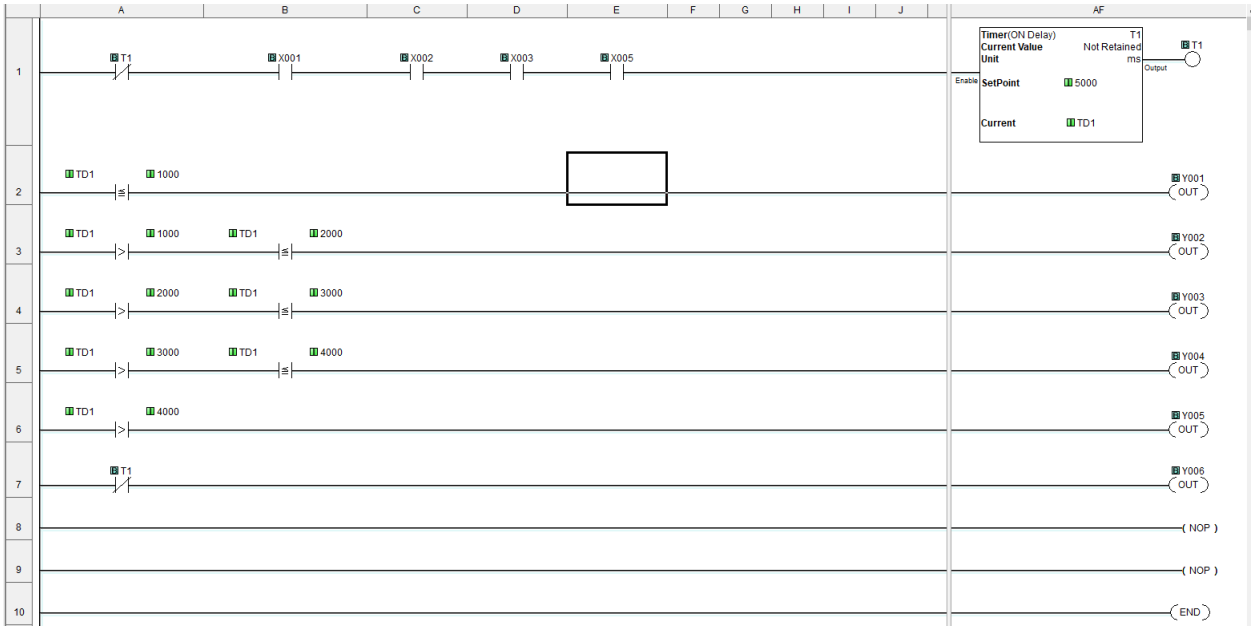


Figure 17 : The correct version of ladder logic

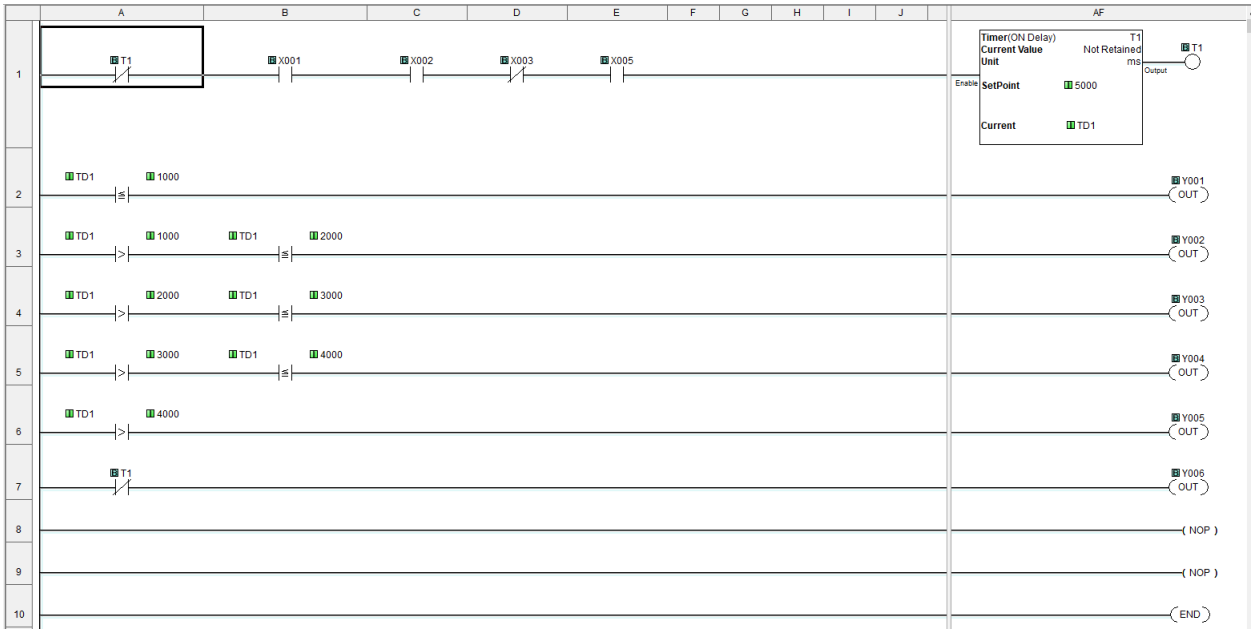


Figure 18 : The altered version of ladder logic

The ladder logic shown in fig-17 was implemented in the PLC. Here, X001, X002, X003, and X004 are inputs to the digital port. Y001, Y002, Y003, Y004, Y005, and Y006 are outputs to the digital port. T1 is the timer, and TD1 represents the timer count. Fig-17 shows the correct

version of the ladder logic. It uses AND conditions for X001, X002, X003, and X004 to turn on the timer. Upon achieving the correct condition, the code executs a blinking LED pattern on the PLC where the output ports Y001 from Y005 are turned on in a circular pattern for one second each. The duration is determined from timer T1. The condition can be expressed in the Boolean form as *X001 & X002 & X003 & X004*. Hence the correct condition to turn on the blinking pattern is X001 from X004 being TRUE.

Fig-18 is the altered version of the ladder logic shown in fig-17. The condition to turn on the timer is altered to *X001 & X002 & !X003 & X004*. Hence the correct condition to turn on the blinking pattern is X001 from X004 being TRUE, TRUE, FALSE, and TRUE respectively. The altered code simulated the case of alteration of the code caused by the cyber intrusion.

Table-2 shown below represents the blinking pattern in a tabular form.

Timer Value TD1 (Milliseconds)	**Y001**	**Y002**	**Y003**	**Y004**	**Y005**
0 - 1000	ON	OFF	OFF	OFF	OFF
1001 - 2000	OFF	ON	OFF	OFF	OFF
2001 - 3000	OFF	OFF	ON	OFF	OFF
3001 - 4000	OFF	OFF	OFF	ON	OFF
4001 - 5000	OFF	OFF	OFF	OFF	ON

Table 2: The Blinking pattern

Digital values obtained from the execution of the ladder logic shown in fig-17 were used to train the neural network, and the results were tested using the ladder logic shown in fig-18 to detect the malfunction caused from an altered code.

4.2.2 GATHERING TRAINING DATA:

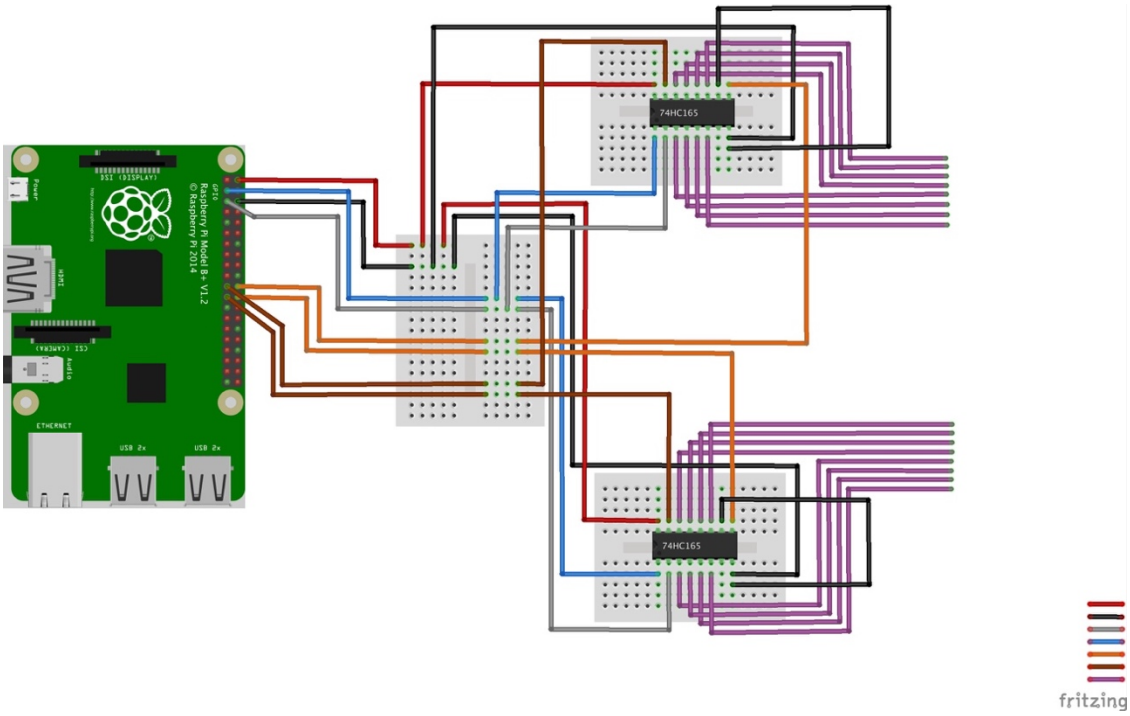


Figure 19: Shift registers setup with Raspberry Pi

Fig-19 shows the setup of shift registers with the Raspberry Pi. This setup was used with the driver program implemented in the Raspberry Pi. The diagram follows the standard color coding, and it is explained in the table below. The driver program is posted in Appendix-A.

Wire Color	Representation
Red	VCC (Power supply)
Black	Ground/Common
Gray	SH/LD
Blue	CLK
Orange	Serial Data QH
Brown	CLK INH (Chip enable CE)
Purple	Parallel Load (A to H)

Table 3: Wire color coding description

The parallel load was connected to the PLC through LM2596 buck converters so that the shift registers can get the parallel data from the PLC's port in 5V signals. The driver program stored digital values gathered from the port in text files named all.txt, input.txt, and output.txt.

These files were imported into an Excel sheet for the MATLAB to process. Fig-20 shows an example of the data gathered.

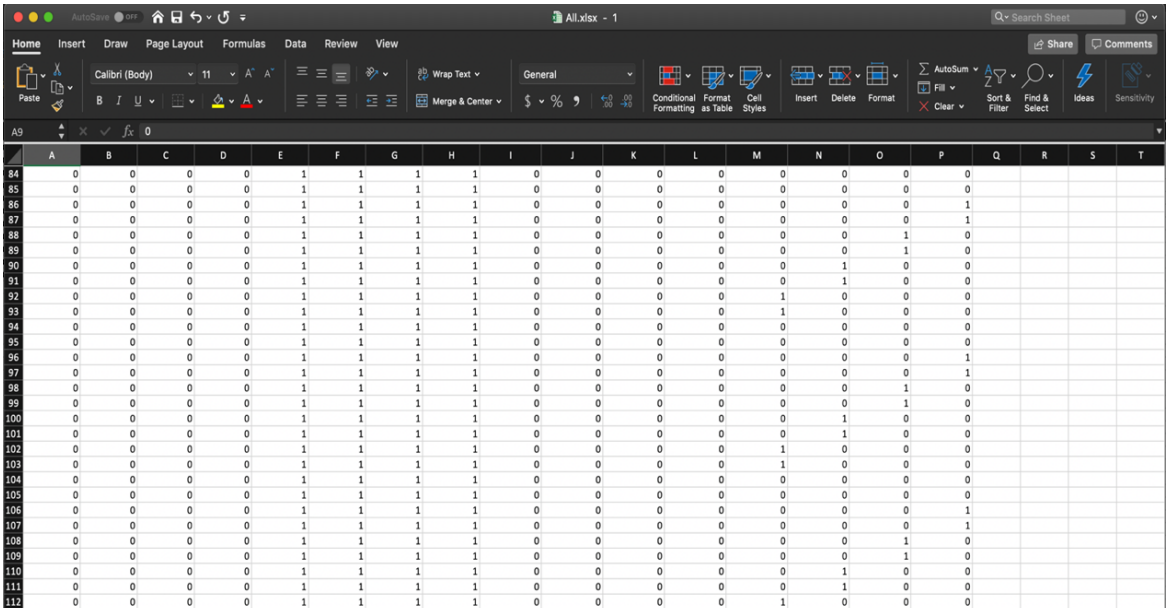


Figure 20: Example of the data gathered using shift registers

4.2.3 TRAINING THE NEURAL NETWORK:

To implement, train, and test the neural network, MATLAB's neural network fitting tool (nftool) was used. The simple feedforward network with 25 neurons in the hidden layer was implemented and trained using the Levenberg-Marquardt algorithm (LM) and Scaled conjugate gradient (SCG).

Inputs and manually added timer states, were given to the neural network's input, and output values of the port were given as the targeted output to the neural network as shown in fig-22.

MATLAB's neural network fitting tool was executed with the command 'nftool' in MATLAB's command window. Upon executing this command, MATLAB launched the neural network fitting tool as shown in fig-21.

Following series of figures shows the training of the neural network using nftool.

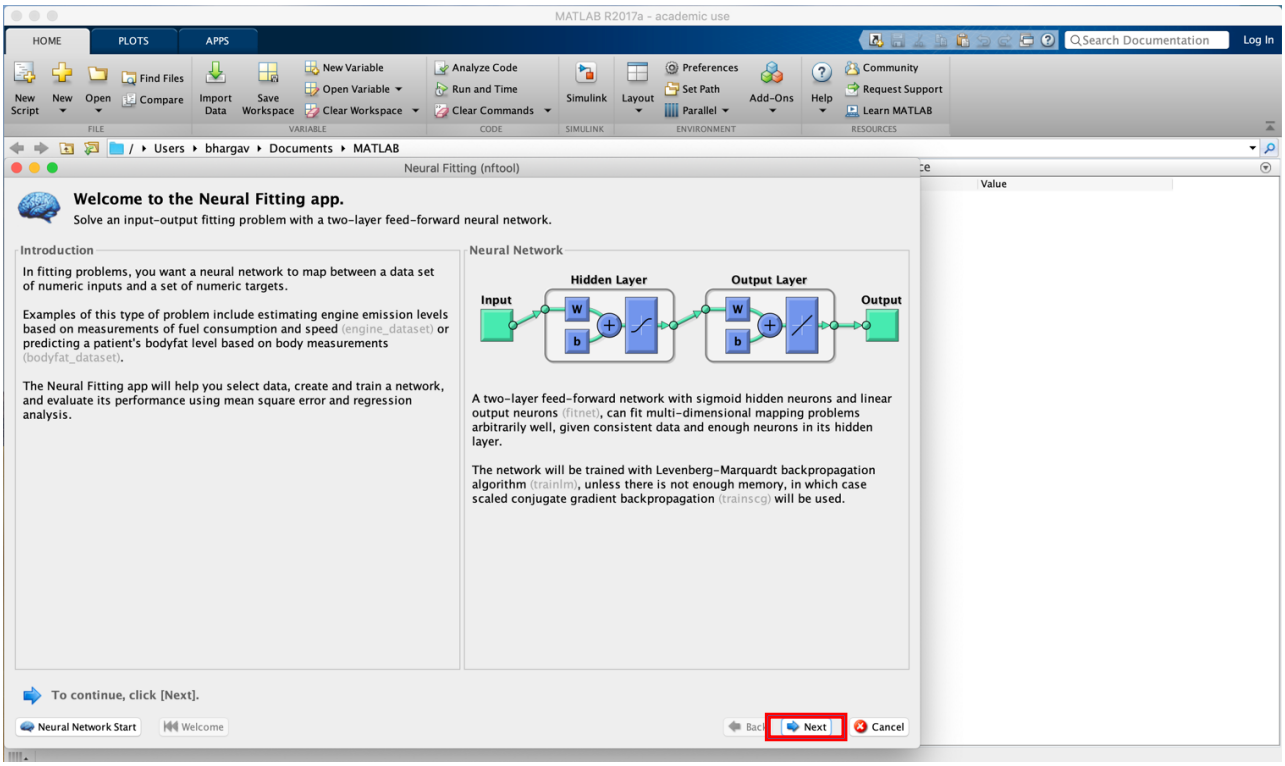


Figure 21: The nftool introduction window

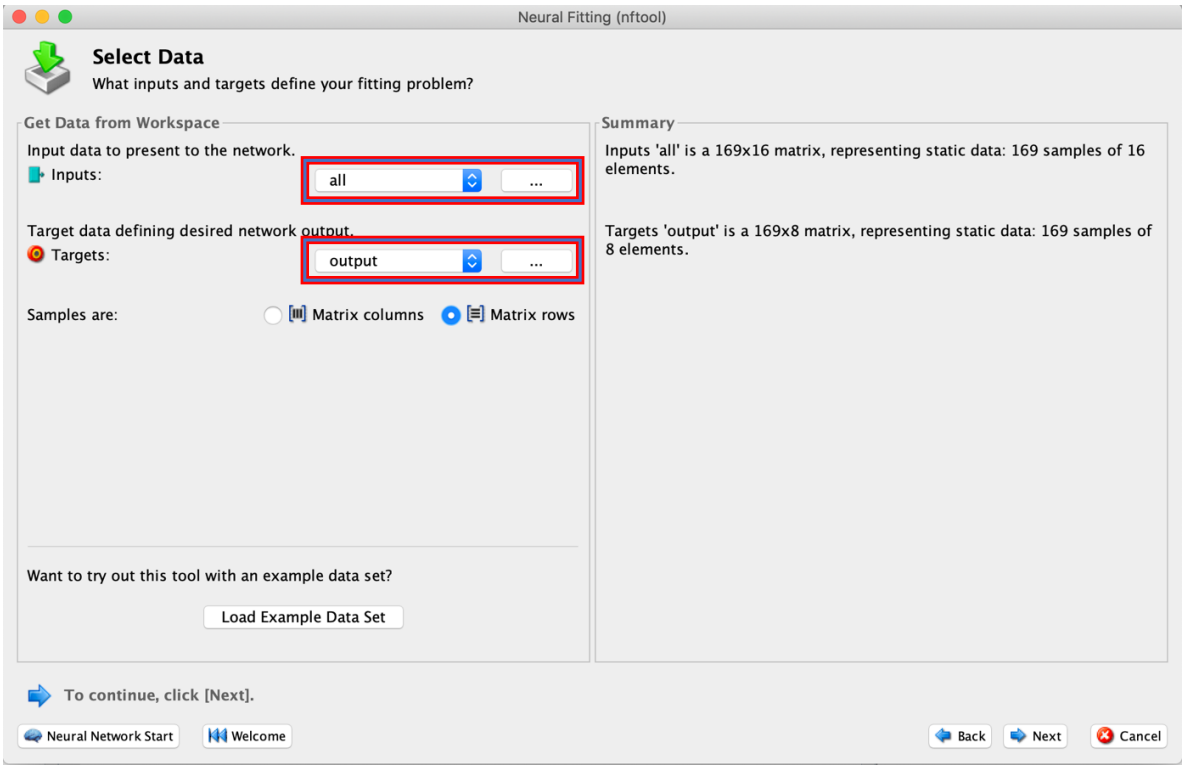


Figure 22: Select inputs and targeted outputs to train the network

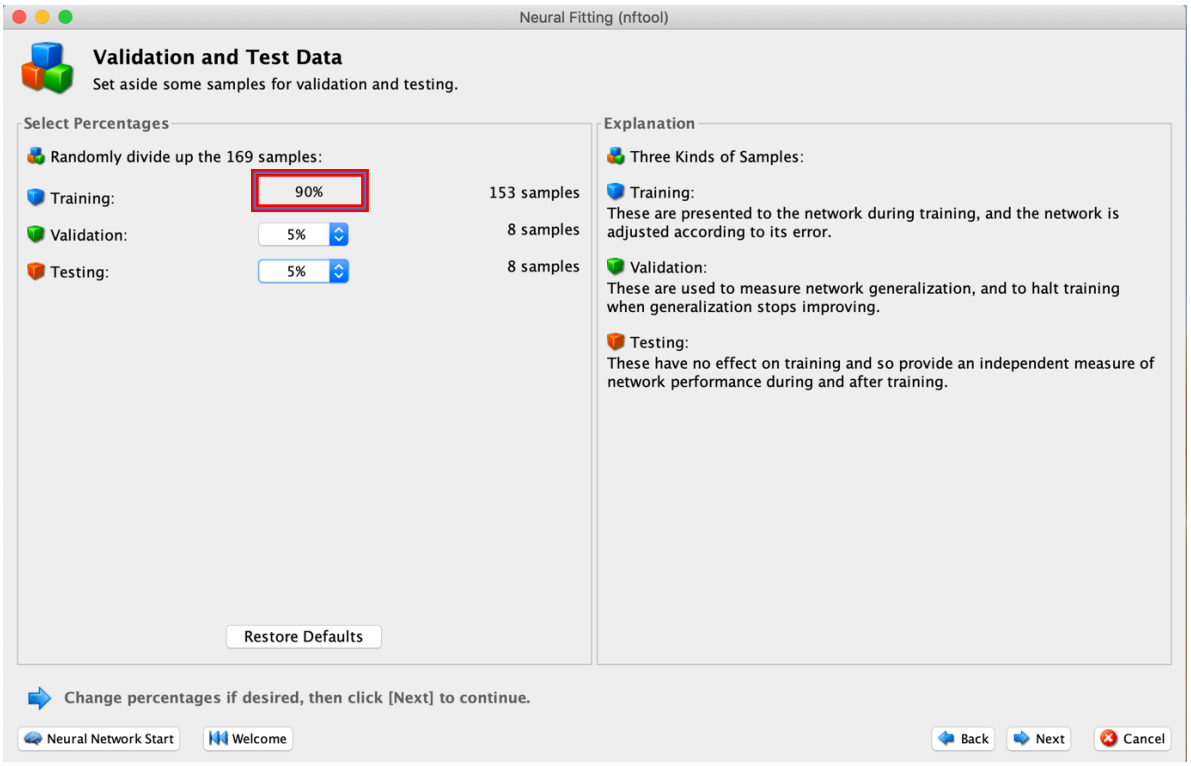


Figure 23: Setup the data distribution for training

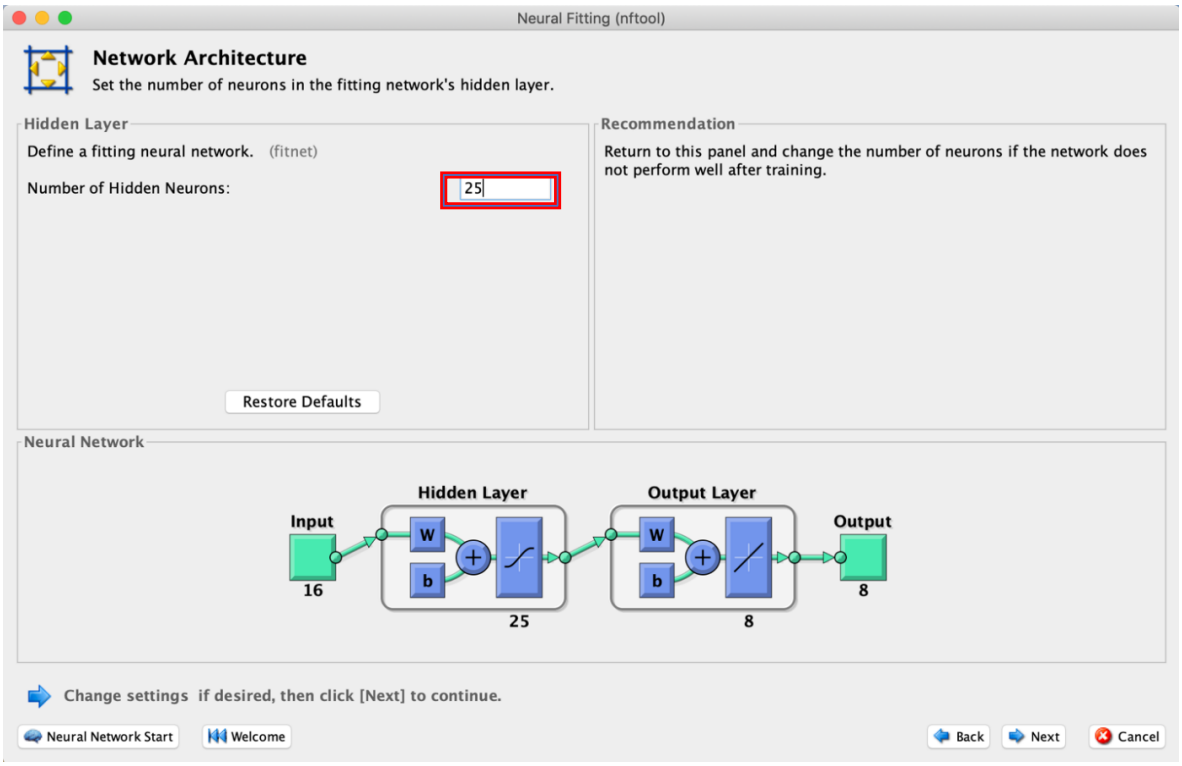


Figure 24: Select the number of neurons in the hidden layer

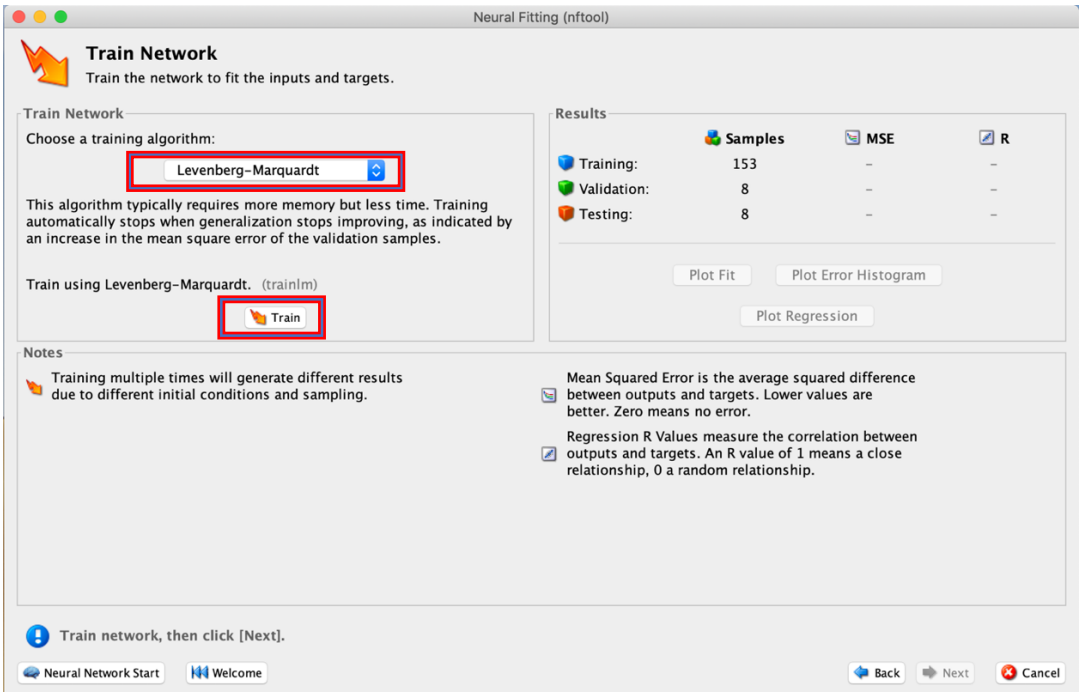


Figure 25: Choose a training algorithm

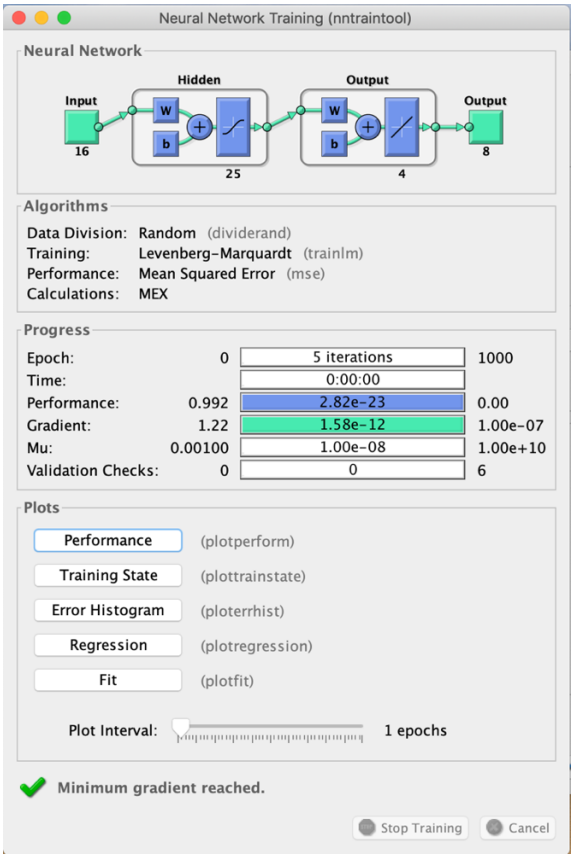


Figure 26: Training performance of neural network using Levenberg-Marquardt algorithm

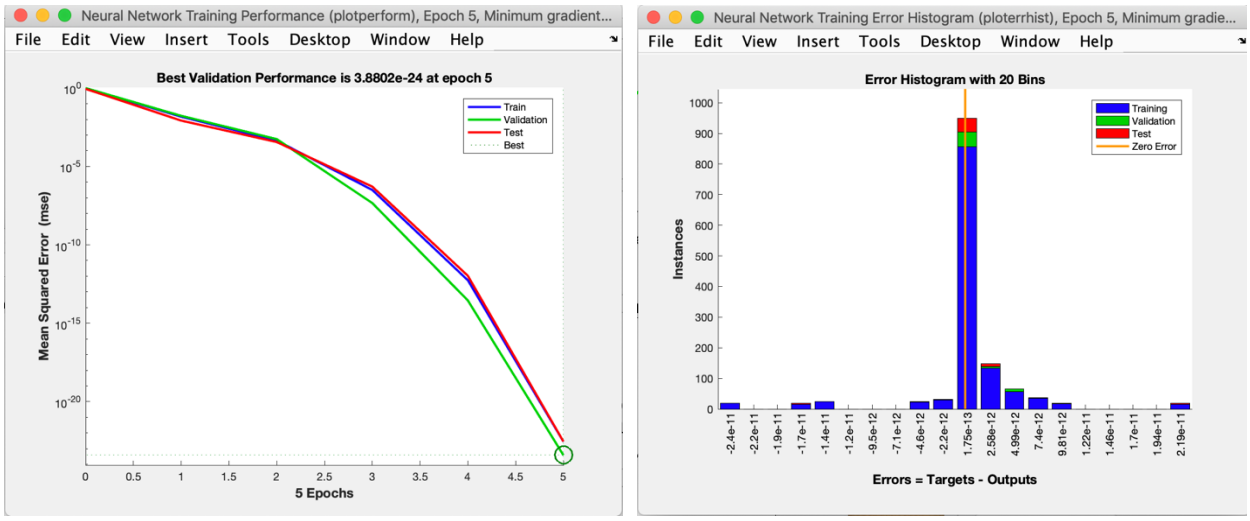


Figure 27: Training performance result plots

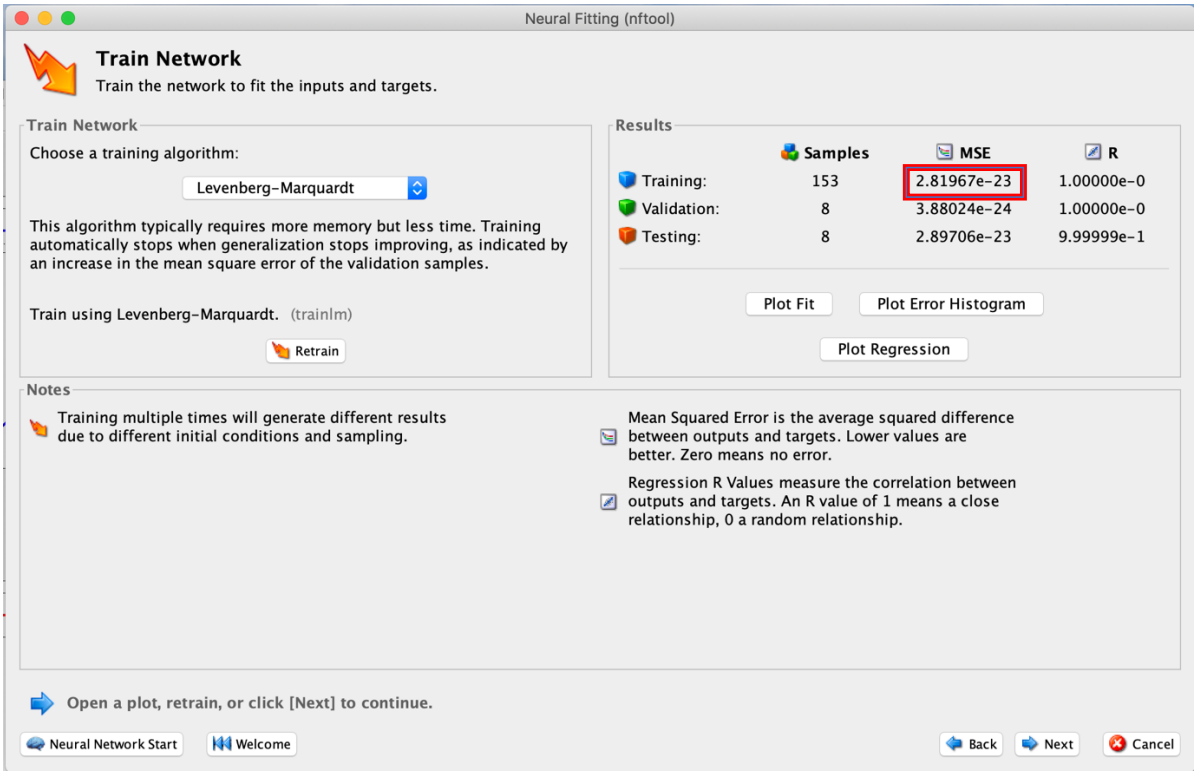


Figure 28: Final training error

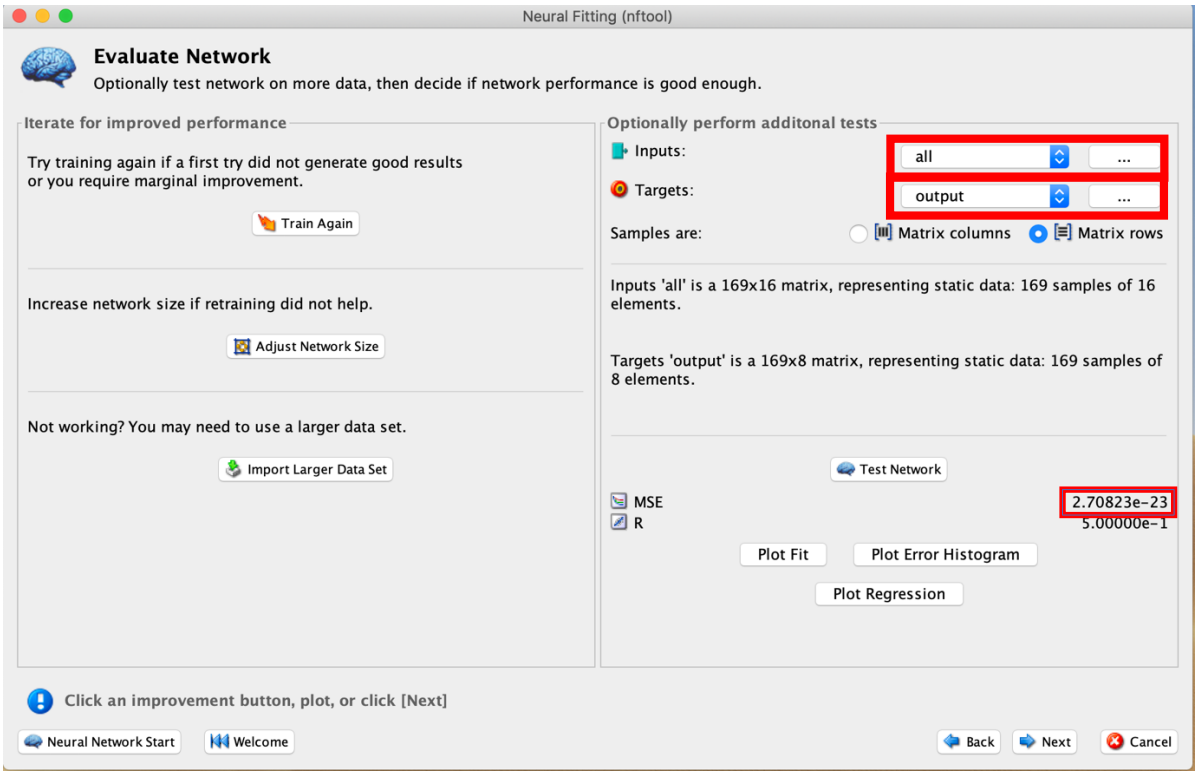


Figure 29: Testing error obtained from correct IO data

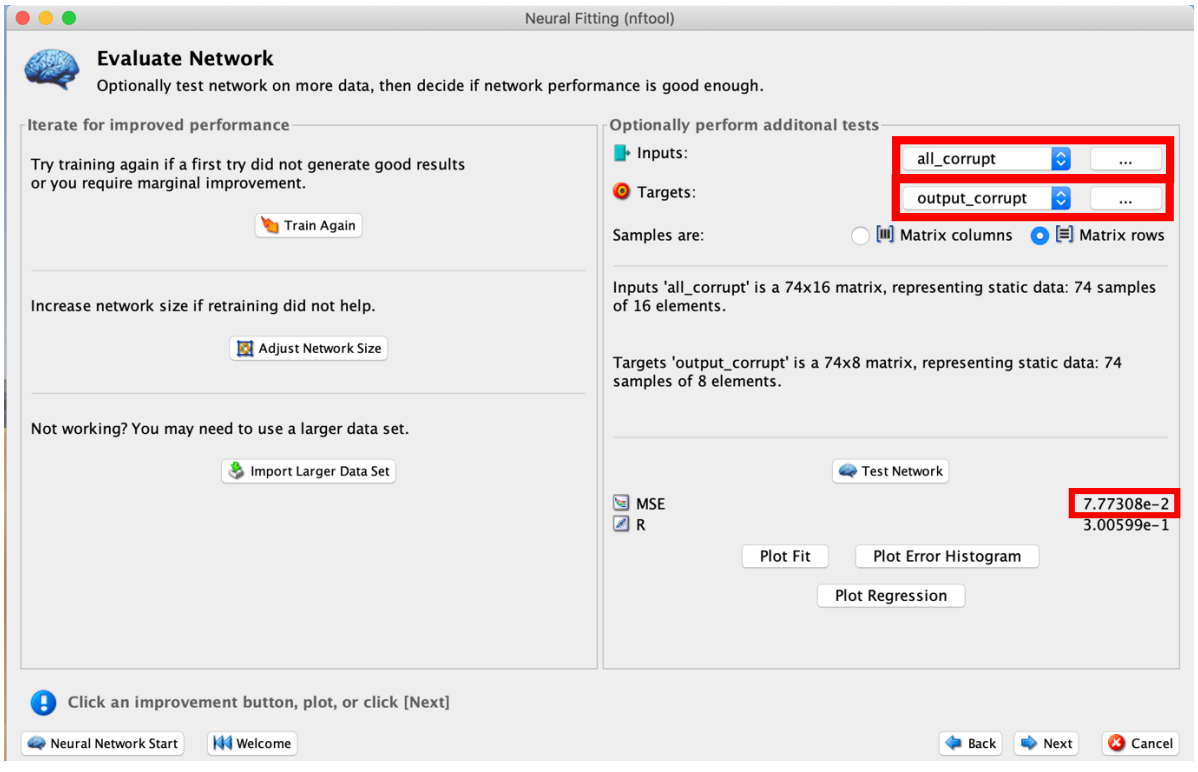


Figure 30: Testing error obtained from corrupt IO data

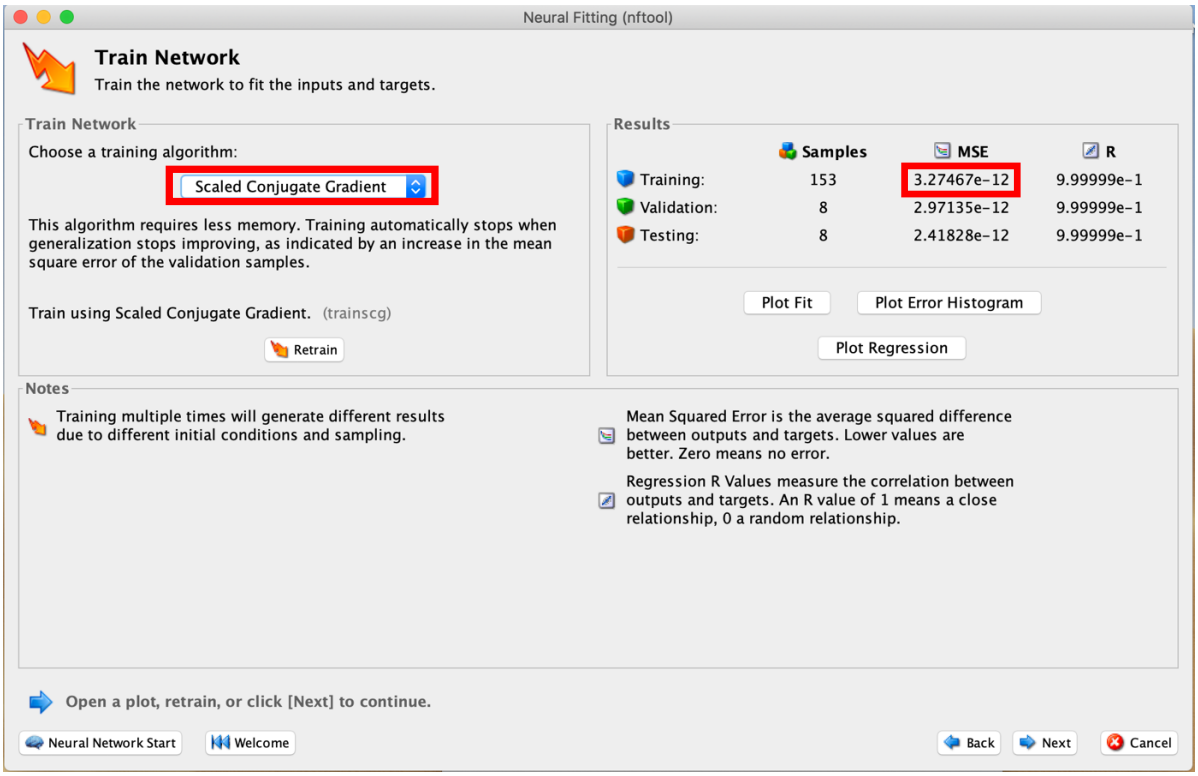


Figure 31: Final training error with Scaled Conjugate Gradient algorithm

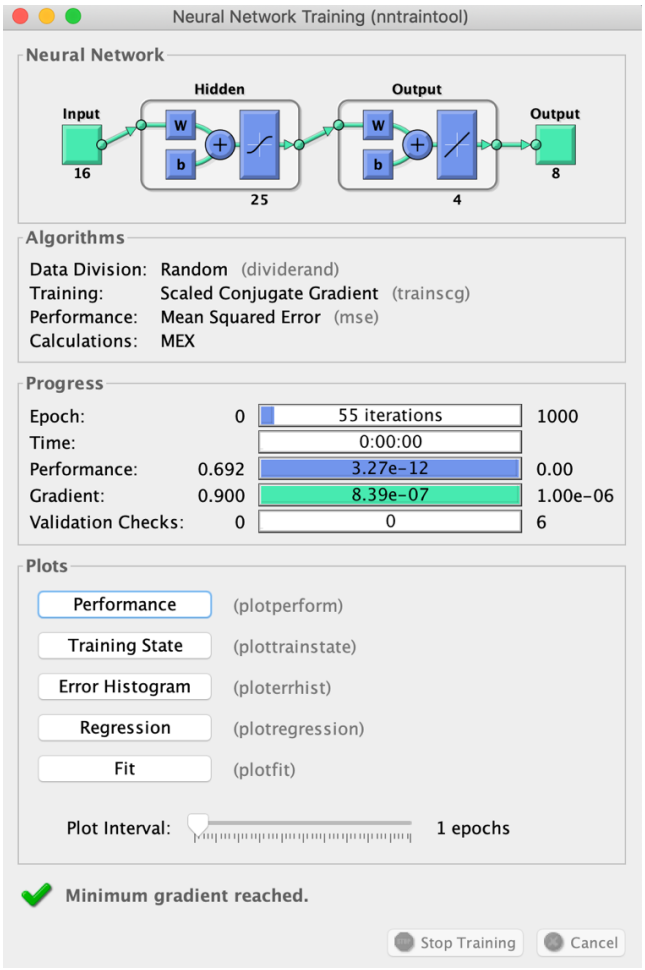


Figure 32: Training performance of neural network using Scaled Conjugate Gradient

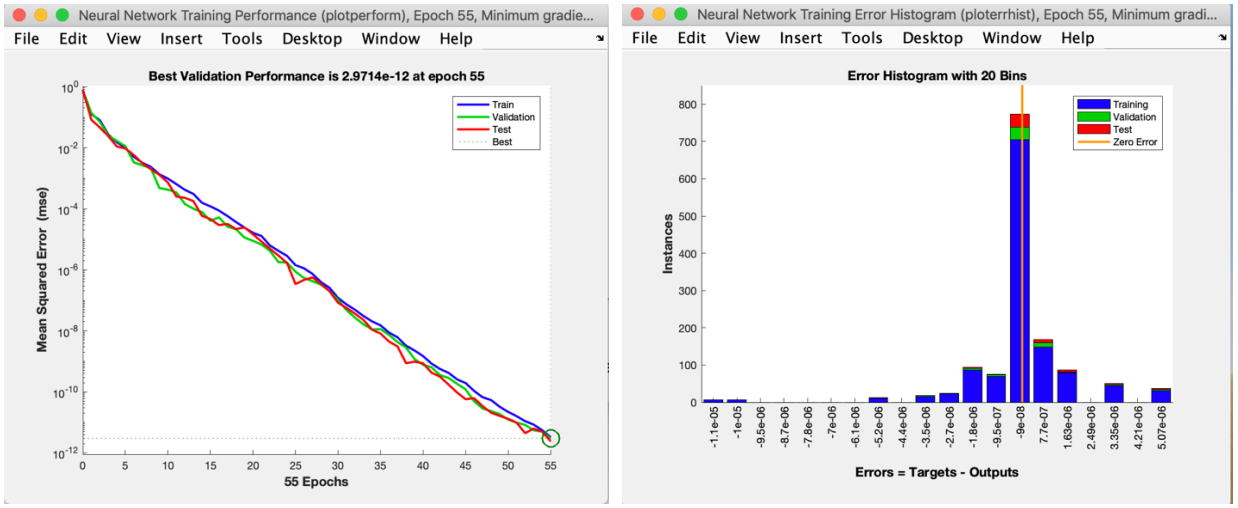


Figure 33: Training performance result plots

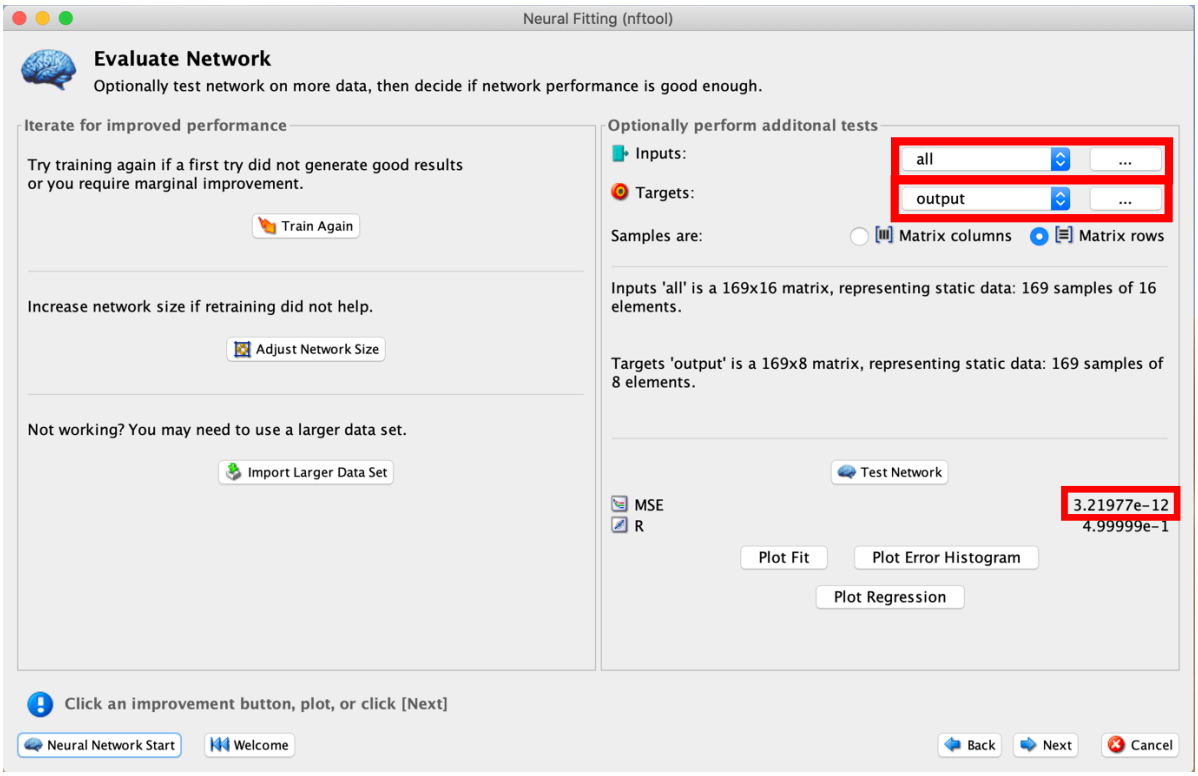


Figure 34: Testing error obtained from correct IO data

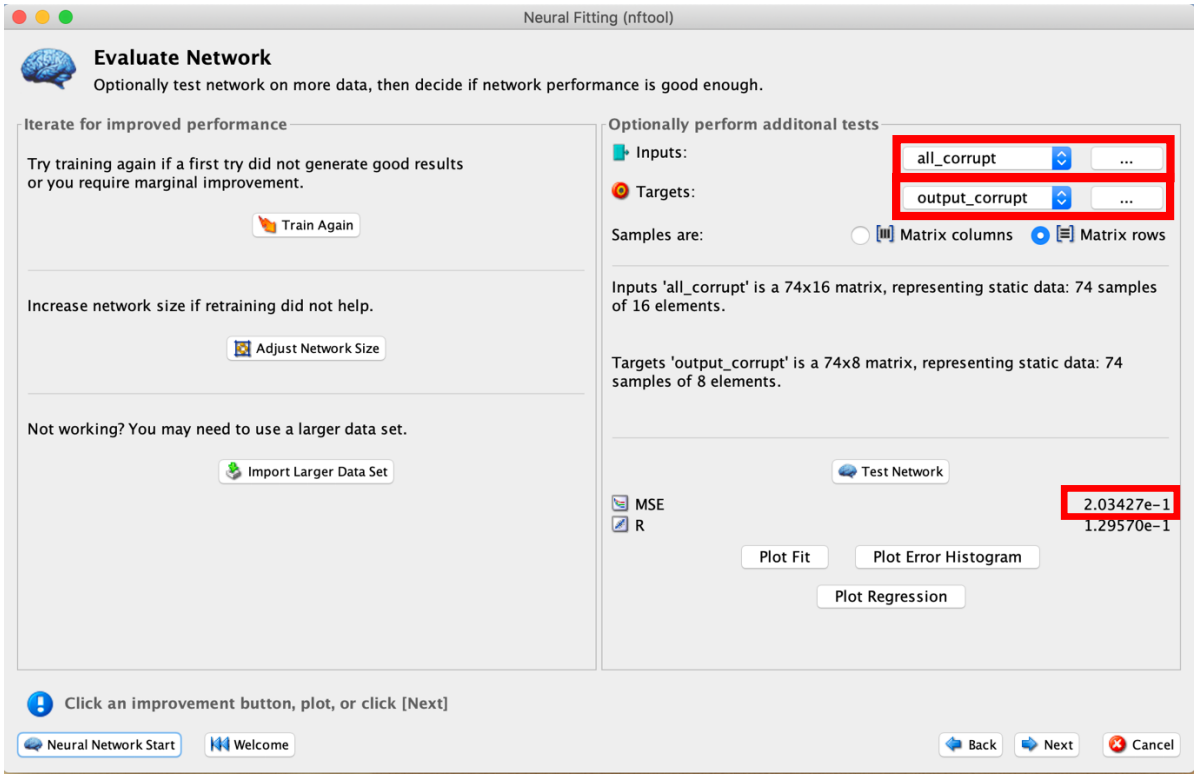


Figure 35: Testing error obtained from corrupt IO data

The IO data from the PLC's port was collected using the shift registers. The correct IO data was harvested by executing the first version of the ladder logic shown in fig-17, and the corrupt IO data was harvested by executing the altered version of the ladder logic shown in fig-18. Also, IO port X001, X002, X003, X004, Y001, Y002, Y003, and Y004 were gathered. Y005 was part of the blinking pattern but it was not possible to include it in the training data set since the external Y005 port is used for supplying power to the output port and cannot be used externally to output data.

The significant difference in MSE (or testing error) was noticeable when comparing the testing error obtained with the correct IO data (fig-29) and the corrupt IO data (fig-30), using the neural network trained with the Levenberg-Marquardt algorithm.

Similar results were obtained when Conjugate Scaled Gradient was used to train the neural network, which can be observed in fig-34 and fig-35.

4.2.4 TESTING NNPLC WITH A PREDICTION MODEL:

Two prediction models were described in chapter-3 section 3.5, named as dynamically deployed, and statically deployed. For better testing and validation, the solution was validated using the statically deployed prediction model in which the neural network's mean squared testing errors were recorded at specific time intervals. Also, this model required the computation of E_p, which was calculated by feeding randomly created subsets of the training data set to the neural network and averaging testing errors of each subset that showed the testing error greater in value than the value of the training error. The training data set and subsets used for this validation are posted in Appendix-B. Corrupt data sets created from executing an altered version of the ladder logic are posted in Appendix-C. The same ladder logic that was used in the previous validation in section 4.2.3 was also used for this validation.

The neural network was trained using Levenberg-Marquardt and Scaled Conjugate Gradient algorithms by following the same procedure described in section 4.2.3. The only difference was that all data sets created were used in the testing window (fig-30 or fig-35) of nftool to generate MSE (mean squared error, also referenced as the testing error) for each data set.

Fig-36 and fig-37 show results obtained from the experiment performed.

	A	B	C	D	E	F	G	H	I	J	K
1			Levenberg-Marquardt					Scaled Conjugate Gradient			
2		Training Error	RMS	R	Cumulative RMS	Plotpoints		RMS	R	Cumulative RMS	Plotpoints
3		Whole test set	2.45E-24	0.999999	-	-		4.23E-12	9.99E-01	-	-
4											
5	Test #	Testing Error									
6	1	Test Set 1	2.10E-24	5.00E-01	2.10E-24	2.10E-24		2.94E-12	5.00E-01	2.94E-12	2.94E-12
7	2	Test Set 2	3.30E-24	5.00E-01	5.40E-24	2.70E-24		2.31E-12	5.00E-01	5.25E-12	2.63E-12
8	3	Test Set 3	2.78E-24	3.75E-01	8.18E-24	2.73E-24		5.87E-12	3.75E-01	1.11E-11	3.71E-12
9	4	Test Set 4	3.20E-24	4.99E-01	1.14E-23	2.84E-24		5.63E-12	5.00E-01	1.68E-11	4.19E-12
10	5	Test Set 5	1.61E-24	3.75E-01	1.30E-23	2.60E-24		2.56E-12	3.75E-01	1.93E-11	3.86E-12
11	6	Test Set 6	3.12E-24	3.75E-01	1.61E-23	2.68E-24		2.64E-12	3.75E-01	2.19E-11	3.66E-12
12	7	Test Set 7	2.38E-24	3.75E-01	1.85E-23	2.64E-24		2.74E-12	3.75E-01	2.47E-11	3.53E-12
13	8	Test Set 8	1.97E-24	5.00E-01	2.05E-23	2.56E-24		2.06E-12	5.00E-01	2.68E-11	3.34E-12
14	9	Test Set 9	3.08E-24	3.75E-01	2.35E-23	2.61E-24		7.09E-12	3.75E-01	3.38E-11	3.76E-12
15	10	Test Set 10	3.14E-24	3.75E-01	2.67E-23	2.67E-24		3.03E-12	3.75E-01	3.69E-11	3.69E-12
16	11	Test Set 11	3.30E-24	5.00E-01	3.00E-23	2.72E-24		7.09E-12	3.75E-01	4.40E-11	4.00E-12
17	12	Test Set 12	3.30E-24	5.00E-01	3.33E-23	2.77E-24		7.09E-12	3.75E-01	5.10E-11	4.25E-12
18	13	Test Set 13	3.30E-24	5.00E-01	3.66E-23	2.81E-24		7.09E-12	3.75E-01	5.81E-11	4.47E-12
19	14	Test Set 14	3.30E-24	5.00E-01	3.99E-23	2.85E-24		7.09E-12	3.75E-01	6.52E-11	4.66E-12
20	15	Test Set 15	3.30E-24	5.00E-01	-	3.30E-24		7.09E-12	3.75E-01	-	7.09E-12
21											
22		Ep	3.00E-24					6.20E-12			

Figure 36: Results obtained from the correct IO test data sets

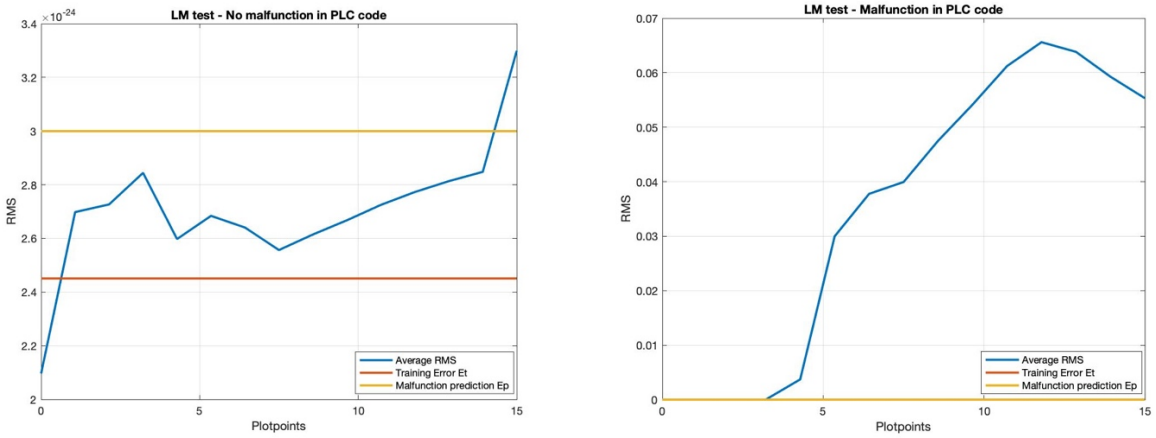
	A	B	C	D	E	F	G	H	I	J	K
1			Levenberg-Marquardt					Scaled Conjugate Gradient			
2	Test #	Testing Error	RMS	R	Cumulative RMS	Cumulative RMS average		RMS	R	Cumulative RMS	Cumulative RMS average
3	1	Test Point 1	2.92E-26	0.00E+00	2.92E-26	2.92E-26		2.56E-12	0.00E+00	2.56E-12	2.56E-12
4	2	Test Point 2	2.92E-26	0.00E+00	5.83E-26	2.92E-26		2.56E-12	0.00E+00	5.11E-12	2.56E-12
5	3	Test Point 3	3.67E-26	0.00E+00	9.50E-26	3.17E-26		5.31E-12	0.00E+00	1.04E-11	3.47E-12
6	4	Test Point 4	7.16E-25	0.00E+00	8.11E-25	2.03E-25		5.84E-12	0.00E+00	1.63E-11	4.06E-12
7	5	Test Point 5	1.86E-02	0.00E+00	1.86E-02	3.71E-03		6.25E-02	0.00E+00	6.25E-02	1.25E-02
8	6	Test Point 6	1.61E-01	2.64E-01	1.80E-01	3.00E-02		3.28E-01	3.03E-01	3.91E-01	6.51E-02
9	7	Test Point 7	8.46E-02	2.13E-01	2.64E-01	3.78E-02		6.43E-01	9.85E-02	1.03E+00	1.48E-01
10	8	Test Point 8	5.51E-02	1.08E-01	3.20E-01	3.99E-02		1.33E-01	1.21E-01	1.17E+00	1.46E-01
11	9	Test Point 9	1.08E-01	2.32E-01	4.28E-01	4.75E-02		2.53E-01	1.76E-01	1.42E+00	1.58E-01
12	10	Test Point 10	1.14E-01	1.10E-01	5.41E-01	5.41E-02		6.45E-01	2.04E-01	2.06E+00	2.06E-01
13	11	Test Point 11	1.32E-01	2.07E-01	6.73E-01	6.12E-02		3.27E-01	1.56E-01	2.39E+00	2.17E-01
14	12	Test Point 12	1.14E-01	1.10E-01	7.87E-01	6.56E-02		6.45E-01	2.04E-01	3.04E+00	2.53E-01
15	13	Test Point 13	4.25E-02	1.25E-01	8.30E-01	6.38E-02		1.36E-01	7.78E-02	3.17E+00	2.44E-01
16	14	Test Point 14	2.43E-24	0.00E+00	8.30E-01	5.93E-02		1.20E-12	0.00E+00	3.17E+00	2.27E-01
17	15	Test Point 15	2.43E-24	0.00E+00	8.30E-01	5.53E-02		1.20E-12	0.00E+00	3.17E+00	2.11E-01

Figure 37: Results obtained from the corrupt IO test data sets

Fig-36 shows the results of the neural network's training using the Levenberg-Marquardt and the Scaled Conjugate Gradient algorithms. We were not concerned with the R values since

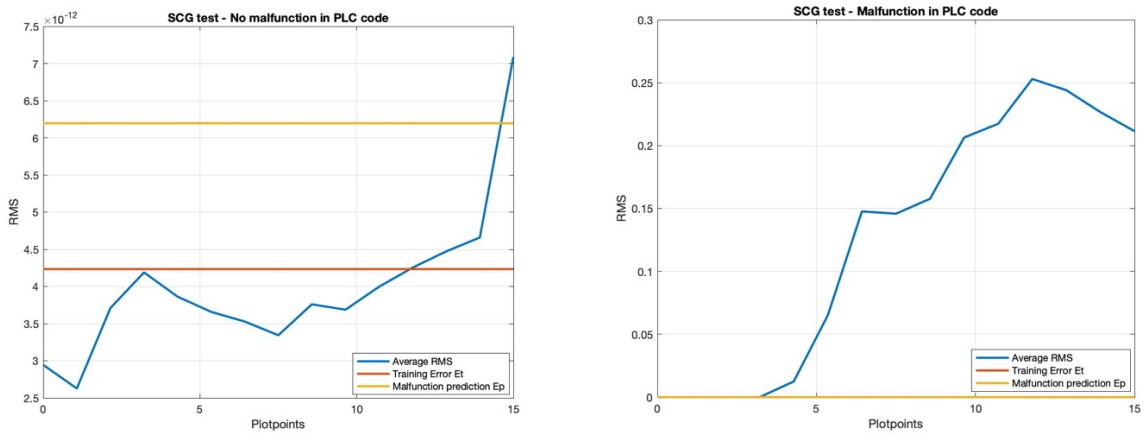
the regression performance of the neural network was not necessary for the experiment. A total of 15 subsets were created from the training data set to calculate E_p. The Root mean squared values of errors from each subset were recorded. The values in green in fig-36 are testing errors that were greater in value than the training error, which were later averaged to calculate E_p. The values in red are testing errors less than the training errors that were neglected from the E_p calculation. The values in yellow are dummy sets of training patterns that showed the least training performance. These yellow values were used just for the plotting purposes. These data sets can also be considered as test sets for ideal PLC operation since they contain the data from the ideal execution of the ladder logic in the PLC.

Fig-37 shows testing errors obtained from the test data sets that were created by executing the altered version of ladder logic. Five input-output patterns were recorded in each data set. The plotting points were created by averaging the cumulative RMS in both figures.



(a) **(b)**

Figure 38: Average cumulative RMS plots (LM)



(a) **(b)**

Figure 39: Average cumulative RMS plot (SCG)

Fig-38(a) Shows the RMS error plot for the test data sets that contain the digital data from the correct version of ladder logic and fig-38(b) shows the same for the altered version of ladder logic while the neural network was trained using LM algorithm. Plot (a) shows that when the PLC had shown no malfunction, the average RMS stayed below E_p and near the training error. Plot (b) shows that when the PLC had executed the altered code, the RMS values went way above the training error and E_p. Hence, the malfunction in the PLC was detected.

Fig-39(a) Shows the RMS error plot for the test data sets that contain the digital data from the correct version of ladder logic and fig-39(b) shows the same for the altered version of ladder logic while the neural network was trained using Conjugate Scaled Gradient algorithm. They show similar results as fig-38(a) and fig-38(b).

The values of errors stayed below the training error for the most part in fig-39(a) compared to fig-38(a) which leads us to a conclusion that SCG algorithm showed well-distributed training for all the patterns in the training data set while LM showed that some of the patterns received better training than other patterns in the training data set. The training error obtained

using LM was lesser than that of SCG. LM showed better overall training, but SCG showed better training distribution for each pattern, meaning that every pattern in the data set received equal training in the case of SCG.

These plots were created using a MATLAB script, which is posted in Appendix-A. The values of each column from spreadsheets shown in fig-36 and fig-37 were imported into MATLAB workspace as numeric matrices for the MATLAB script to work.

5. CHAPTER-5: CONCLUSION AND FUTURE WORK

5.1 CONCLUSION:

PLC is called the heart of any automation system, and they are widely used in the industry. Because of their wide range of applications and usage; improvements to these controllers are always beneficial. We introduced the concept to implement machine learning onto PLCs, which opens many doors of improvements in PLC driven automation systems. One of the most important improvements is better fault detection. After an automation system is designed, a neural network can be trained to learn the ideal values of inputs and outputs to the PLC during the simulation of an automated system before it is implemented at the customer's site. The neural network can then be trained further at the customer's site to learn the characteristics of "everyday" operations of the automation system. Once the fully trained neural network is deployed to run in parallel with the PLC, it can monitor the activities on the PLC's ports and look for any abnormal values that wouldn't occur during the normal "everyday" operation. These abnormal values may result from a malfunction of edge devices like sensors, motors, valves, etc.; cyber intrusions, bugs in the PLC code, and henceforth. Since the neural network has learned the ideal characteristics and the behavior of the automation system, it can detect if it behaves abnormally. For example, if the PLC starts executing error handlings codes that are usually not executed during the "everyday" operation, the neural network's predictions

can alert the operators of the system that the code is not executing the way how it should be on an everyday basis.

It is possible that the system could run into faults that were not factored-in during the development of the code since there are many possibilities of what could go wrong. Error handling codes for some of those possibilities are often overlooked due to human errors and limited understanding of the problem. Thus it may cause the whole system to fail since it lacked error handling for such errors. The neural network can detect abnormal behaviors, and its predictions can save the system from the failure for such cases that could have a significant financial impact or put lives at risk.

The successful validation of the hypothesis shows that, with more efforts and research, the NNPLC concept can become a practical application for real automation systems.

5.2 FUTURE WORK:

As mentioned in the validation, the concept was tested on a PLC with only digital IO ports. Since it was not possible to harvest the timer values at the moment due to the restrictions of PLC programming software, this concept was not validated for complex PLC codes implemented in real automation systems whose logic ineluctably rely on timers and analog ports of the PLC. The solution to extract timer values needs more researching.

Better suited neural network architectures for complex ladder logics, and training algorithms for the NNPLC application can be researched and developed.

More efforts are needed to implement the dynamic deployment of the neural network discussed in chapter-3 since it requires synchronization between the PLC and the neural network.

The application-specific neural network code should be written as per the requirements of the automation system rather than using available general-purpose software to implement neural network, i.e., MATLAB, NeuroSolutions, etc., in order to achieve better validation, optimization, and to implement this concept into real systems. Live plots of difference between predictions and actual outputs needs to be implemented in order to monitor the system in real time.

REFERENCES

[1] B.K. Bose. 2009. Power Electronics and Motor Drives Recent Progress and Perspective. *IEEE Trans. Ind. Electron.* 56, 2 (February 2009), 581–588. DOI:https://doi.org/10.1109/TIE.2008.2002726

[2] Deep AI inc. 2018. Neural Network Definition | DeepAI. *https://deepai.org*. Retrieved March 7, 2019 from https://deepai.org/machine-learning-glossary-and-terms/neural-network

[3] R. (Roger) Fletcher. 1987. *Practical methods of optimization*. Wiley.

[4] N Fnaiech, Farhat Fnaiech, B W Jervis, and Mohamed Cheriet. 2009. *The combined statistical stepwise and iterative neural network pruning algorithm*.

[5] Martin Fodslette Møller. 1993. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks* 6, (1993), 525–533.

[6] Georgel Gabor, Cosmin Pintilie, Catalin Dumitrescu, Nituca Costica, and Adrian Traian Plesca. 2018. Application of Industrial PROFIBUS-DP Protocol. In *EPE 2018 - Proceedings of the 2018 10th International Conference and Expositions on Electrical And Power Engineering*, 614–617. DOI:https://doi.org/10.1109/ICEPE.2018.8559857

[7] Umut Güçlü and Marcel A. J. van Gerven. 2017. Modeling the Dynamics of Human Brain Activity with Recurrent Neural Networks. *Front. Comput. Neurosci.* 11, (February 2017), 7. DOI:https://doi.org/10.3389/fncom.2017.00007

[8] Shuxiang Guo, Juan Du, Xiufen Ye, Hongtao Gao, and Yizhou Gu. 2010. Real-time adjusting control method based on attitude sensor signal feedback and its application in spherical underwater vehicle. *2010 IEEE Int. Conf. Inf. Autom. ICIA 2010* (2010), 1393–1397. DOI:https://doi.org/10.1109/ICINFA.2010.5512098

[9] M.T. Hagan and M.B. Menhaj. 1994. Training feedforward networks with the Marquardt algorithm. *IEEE Trans. Neural Networks* 5, 6 (1994), 989–993. DOI:https://doi.org/10.1109/72.329697

[10] Bhargav Joshi and Bhavin Patel. 2016. Boiler Automation and Turbine Protection At By. Dharmsinh Desai University, Nadiad, India.

[11] Murat Kayri. 2016. Predictive Abilities of Bayesian Regularization and Levenberg–Marquardt Algorithms in Artificial Neural Networks: A Comparative Empirical Study on Social Data. *Math. Comput. Appl.* 21, 2 (May 2016), 20. DOI:https://doi.org/10.3390/mca21020020

[12] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. 2015. *Artificial Neural Networks as Models of Neural Information Processing*. DOI:https://doi.org/10.1038/nature14539

[13] Yadong Li, Danlan Li, Wenqiang Cui, and Rui Zhang. 2011. Research based on OSI model. *2011 IEEE 3rd Int. Conf. Commun. Softw. Networks, ICCSN 2011* (2011), 554–557. DOI:https://doi.org/10.1109/ICCSN.2011.6014631

[14] Albert Malvino and David Bates. 2016. *Electronic Principles*. Retrieved from

http://www.just.edu.jo/CoursesAndLabs/Electronics 1_Phy 231/Syllabus_231.doc
[15] M. Morris Mano and Michael D. Ciletti. 2013. *Digital design : with an introduction to the verilog hdl*. Retrieved from http://www.mypearsonstore.com/bookstore/digital-design-9780132774208
[16] Donald W. Marquardt. 1963. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *J. Soc. Ind. Appl. Math.* 11, 2 (June 1963), 431–441. DOI:https://doi.org/10.1137/0111030
[17] Microchip. 2008. Mcp3008. (2008). Retrieved from https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf
[18] Wael A. Monsef and Fayez FG Areed. 2006. Design of a Neural – PLC controller FOR Industrial Plant. *Proc. 2006 Int. Conf. Mach. Learn.* (2006).
[19] A.K. Palit and D Popovic. 2005. *Neural Networks Approach*. Springer. Retrieved from http://www.springer.com/978-1-85233-948-7
[20] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (October 1986), 533–536. DOI:https://doi.org/10.1038/323533a0
[21] Osvaldo Simeone. 2018. A Very Brief Introduction to Machine Learning with Applications to Communication Systems. *IEEE Trans. Cogn. Commun. Netw.* 4, 4 (2018), 648–664. DOI:https://doi.org/10.1109/TCCN.2018.2881442
[22] Gang-Neng Sung, Ching-Lin Wang, Ping-Chang Jui, and Chua-Chin Wang. 2010. A high-efficiency DC-DC buck converter for sub-3×VDD power supply. In *2010 IEEE International Conference on Integrated Circuit Design and Technology*, 164–167. DOI:https://doi.org/10.1109/ICICDT.2010.5510269
[23] Texas Instruments. 2013. 8-BIT PARELLEL-LOAD SHIFT REGISTERS. *www.ti.com*, 26. Retrieved from http://www.ti.com/product/SN74HC165?qgpn=sn74hc165
[24] D. M. Titterington. 2004. Bayesian Methods for Neural Networks and Related Models. *Stat. Sci.* 19, 1 (February 2004), 128–139. DOI:https://doi.org/10.1214/088342304000000099
[25] B.M. Wilamowski, N.J. Cotton, Okyay Kaynak, and G. Dundar. 2008. Computing Gradient Vector and Jacobian Matrix in Arbitrarily Connected Neural Networks. *IEEE Trans. Ind. Electron.* 55, 10 (October 2008), 3784–3790. DOI:https://doi.org/10.1109/TIE.2008.2003319
[26] B.M. Wilamowski, D. Hunter, and A. Mabnowski. 2004. Solving parity-N problems with feedforward neural networks. (2004), 2546–2551. DOI:https://doi.org/10.1109/ijcnn.2003.1223966
[27] Bogdan Wilamowski. 2011. Neural Network Architectures. (2011), 1–17. DOI:https://doi.org/10.1201/b10604-9
[28] Bogdan M Wilamowski. 2009. How Not to Be Frustrated with Neural Networks. *IEEE Ind. Electron. Mag.* December (2009), 56–63.
[29] 5 algorithms to train a neural network | Neural Designer. Retrieved March 8, 2019 from https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network
[30] Amazon.com: eBoot 6 Pack LM2596 DC to DC Buck Converter 3.0-40V to 1.5-35V Power Supply Step Down Module: Automotive. Retrieved March 19, 2019 from https://www.amazon.com/eBoot-LM2596-Converter-3-0-40V-1-5-35V/dp/B01GJ0SC2C
[31] MCP3008 | Raspberry Pi Analog to Digital Converters | Adafruit Learning System.

Retrieved March 25, 2019 from https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/mcp3008
[32] Fit Data with a Shallow Neural Network - MATLAB & Simulink. Retrieved May 1, 2019 from https://www.mathworks.com/help/deeplearning/gs/fit-data-with-a-neural-network.html

APPENDIX-A

A-1 Shift register's driver program for Raspberry Pi (PISO.py):

```
import RPi.GPIO as GPIO
import time
import sys

#-------------------------------------------------------------------------#
#                          Global Definitions                             #
#-------------------------------------------------------------------------#

#SH/LD pin initialized on GPIO pin#3
shld_pin = 3

#Clock pin initialized on GPIO pin#5
clk_pin = 5

#Serial data pin for the first shift register initialized on GPIO pin#22
data_pin = 22

#Serial data pin for the second shift register initialized on GPIO pin#24
data_pin_1 = 24

#Chip enable pin (CLK INH) for the first shift register initialized on GPIO pin#21
ce_pin = 21

#Chip enable pin (CLK INH) for the second shift register initialized on GPIO pin#23
ce_pin_1 = 23

LSB = 0             #LSB to output serial data as the least significant bit first format
MSB = 1             #MSB to output serial data as the most significant bit first format

#-------------------------------------------------------------------------#
#                         Function Definitions                            #
#-------------------------------------------------------------------------#

#Function to initialize GPIO ports on Raspberry Pi
def initialize():
```

```
        GPIO.setmode(GPIO.BOARD)        #Enable GPIO port
        GPIO.setup(7, GPIO.OUT)         #Setup GPIO pin#7 in output mode
        GPIO.setup(shld_pin, GPIO.OUT)  #GPIO for SH/LD pin in output mode
        GPIO.setup(ce_pin, GPIO.OUT)    #GPIO for CLK INH pin in output mode
        GPIO.setup(ce_pin_1,GPIO.OUT)   #GPIO for CLK INH pin in output mode
        GPIO.setup(clk_pin,GPIO.OUT)    #GPIO for CLK pin in output mode
        GPIO.setup(data_pin, GPIO.IN)   #GPIO for serial data pin in input mode
        GPIO.setup(data_pin_1, GPIO.IN) #GPIO for serial data pin in input mode
        GPIO.output(7, 1)               #Set GPIO port#7 to HIGH
        GPIO.output(clk_pin, 1)         #Set GPIO port#5 to HIGH
        GPIO.output(shld_pin, 1)        #Set GPIO port#3 to HIGH
        return;

 #Function to execute the end procedure of the driver
def endprocess():
        print "Cleaning GPIO...",
        GPIO.cleanup()                  #Cleanup GPIO
        print "Done"
        print "Closing files...",
        fh1.close()                     #Close file input.txt
        fh2.close()                     #Close file output.txt
        fh3.close()                     #Close file all.txt
        print "Done"
        print "Syncing data files on google cloud\n"

#Function to format the serial data to be written into the file
#Format example: 0 0 1 0 0 1 0 1
def fmat(var):
        ns = ""
        for i in range(0,8):
                num = (var >> i) & 1
                if(num == 0):
                        num = "0"
                else:
                        num = "1"
                ns += num + " "
        ns = ns[::-1]
        return ns;

#Function to execute the timing diagram of the shift register
#This function operates shift registers to achieve parallel to serial conversion
def read_shift_regs(data,ce):
        the_shifted = 0
        #Set SH/LD pin to LOW to load the parallel data into the shift register
        GPIO.output(shld_pin, 0)
```

```
        #Hold SH/LD's value as LOW for 5 microseconds
        time.sleep(5/1000000.0)

        #Set SH/LD pin to HIGH to shift the loaded parallel data to output serial data
        GPIO.output(shld_pin, 1)

        #Hold SH/LD's value as HIGH for 5 microseconds to initialize the shifting operation
        time.sleep(5/1000000.0)

        #set clock pulse to HIGH
        GPIO.output(clk_pin, 1)

        #Set the CLK INH to HIGH to enable the clock signal to the shift registers
        GPIO.output(ce, 0)

        #Call ShiftIn function to get the serial data from the shift registers
        the_shifted = ShiftIn(data, clk_pin, MSB)

        #Set the CLK INH to LOW to diable the clock signal to the shift registers
        GPIO.output(ce, 1)

        return the_shifted;     #return the shifted value

#Function to perform the shifting operation and output the serial data
def ShiftIn(data, clk, order):
        value = 0
        #Perform eight shifts to output 8-bit serial data
        for i in range(0,8):
        #Shift bits on each asynchronous clock pulse
                GPIO.output(clk, 1)                             #Set the clock signal to HIGH
                if(order == LSB):                               #Check the output order
                        value |= GPIO.input(data) << i          #Serial output in LSB fashion
                else:
                        value |= GPIO.input(data) << (7-i)      #Serial output in MSB fashion
                GPIO.output(clk, 0)                             #Set the clock signal to LOW
        return value;

#-------------------------------------------------------------------------#
#                                 Main                                    #
#-------------------------------------------------------------------------#

initialize()                                                    #Initialize Raspberry Pi
#Open Files to write mode
fh1 = open('/home/pi/AIPLC/data/input.txt',"w")                 #Open input.txt
```

```
fh2 = open('/home/pi/AIPLC/data/output.txt',"w")              #Open output.txt
fh3 = open('/home/pi/AIPLC/data/all.txt',"w")                 #Open all.txt
try:
        while(True):
                val = read_shift_regs(data_pin,ce_pin)        #Read from the shift register-1
                val1 = read_shift_regs(data_pin_1,ce_pin_1)   #Read from the shift register-2
                s1 = fmat(val)                                #Format the read value
                s2 = fmat(val1)                               #Format the read value

                #Write serial values in files
                fh1.write(s1 + '' + "\r\n")
                fh2.write(s2 + '' + "\r\n")
                fh3.write(s1 + ' ' + s2 + "\r\n")

                #Display serial values on terminal
                print(s1),
                print"\t",
                print(val),
                print "\t",
                print(s2),
                print "\t",
                print(val1)
                time.sleep(0.5)
#End execution in case if a keyboard interrupt detected
except (KeyboardInterrupt, SystemExit):
        print "\nKeyboard interrupt detected..."
        endprocess()
except:
        print "Fatal Error"
        endprocess()
```

A-2 MATLAB script to plot error results:

```
clc;
%Create plotpoints for 15 test sets
Plotpoints(:,1) = linspace(0,15,15);

%Plot LM performance for no malfunction in PLC code
figure(1)
plot(Plotpoints,LM_correct_plotpoints)
hold on
plot(Plotpoints,LM_training_error)
plot(Plotpoints,LM_Ep)
hold off
grid
legend('Average RMS','Training Error Et','Malfunction prediction Ep')
title('LM test - No malfunction in PLC code')
xlabel('Plotpoints')
ylabel('RMS')

%Plot LM performance for a malfunction in PLC code
figure(2)
plot(Plotpoints,LM_malfunction_plotpoints)
hold on
plot(Plotpoints,LM_training_error)
plot(Plotpoints,LM_Ep)
hold off
grid
legend('Average RMS','Training Error Et','Malfunction prediction Ep')
title('LM test - Malfunction in PLC code')
xlabel('Plotpoints')
ylabel('RMS')

%Plot SCG performance for no malfunction in PLC code
figure(3)
plot(Plotpoints,SCG_correct_plotpoints)
hold on
plot(Plotpoints,SCG_training_error)
plot(Plotpoints,SCG_Ep)
hold off
grid
legend('Average RMS','Training Error Et','Malfunction prediction Ep')
title('SCG test - No malfunction in PLC code')
xlabel('Plotpoints')
ylabel('RMS')
```

```
%Plot SCG performance for a malfunction in PLC code
figure(4)
plot(Plotpoints,SCG_malfunction_plotpoints)
hold on
plot(Plotpoints,SCG_training_error)
plot(Plotpoints,SCG_Ep)
hold off
grid
legend('Average RMS','Training Error Et','Malfunction prediction Ep')
title('SCG test - Malfunction in PLC code')
xlabel('Plotpoints')
ylabel('RMS')
```

APPENDIX-B

B-1 Complete training data set

Inputs:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0

0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1

0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0

0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0

0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0

B-2 Training subset-1

Inputs:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0

B-3 Training subset-2

Inputs:

0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0

B-4 Training subset-3

Inputs:

0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0

Outputs:

0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0

B-5 Training subset-4

Inputs:

0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1

B-6 Training subset-5

Inputs:

0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

B-7 Training subset-6

Inputs:

0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0

Outputs:

0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0

B-8 Training subset-7

Inputs:

0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

B-9 Training subset-8

Inputs:

0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

B-10 Training subset-9

Inputs:

0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

B-11 Training subset-10

Inputs:

0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0

Outputs:

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

APPENDIX-C

C-1 Test point-1

Inputs:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

C-2 Test point-2

Inputs:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

C-3 Test point-2

Inputs:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

C-4 Test point-2

Inputs:

0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

C-5 Test point-2

Inputs:

0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1

0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

C-6 Test point-2

Inputs:

0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	0	1	1	0	0	0	0	0	1	1	0
0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	0

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0
0	0	0	0	0	1	1	0
0	0	0	0	0	1	0	0

C-7 Test point-2

Inputs:

0	0	0	0	1	0	1	1	0	0	0	0	1	1	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	1	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1

C-8 Test point-2

Inputs:

0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

C-9 Test point-2

Inputs:

0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	0

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0

C-10 Test point-2

Inputs:

0	0	0	0	1	0	1	1	0	0	0	0	0	1	1	0
0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	1	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0

Outputs:

0	0	0	0	0	1	1	0
0	0	0	0	0	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0

C-11 Test point-2

Inputs:

0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	0

Outputs:

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0

C-12 Test point-2

Inputs:

0	0	0	0	1	0	1	1	0	0	0	0	0	1	1	0
0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	1	0	0

0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0

Outputs:

0	0	0	0	0	1	1	0
0	0	0	0	0	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0

C-13 Test point-2

Inputs:

0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

C-14 Test point-2

Inputs:

0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

C-15 Test point-2

Inputs:

0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

C-16 The complete test set

Inputs:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	0	1	1	0	0	0	0	0	1	1	0
0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	1	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	0	1	1	0	0	0	0	0	1	1	0
0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	1	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	1
0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	0
0	0	0	0	1	0	1	1	0	0	0	0	0	1	1	0
0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	1	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1

0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1

Outputs:

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0
0	0	0	0	0	1	1	0
0	0	0	0	0	1	0	0
0	0	0	0	1	1	0	0

0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0
0	0	0	0	0	1	1	0
0	0	0	0	0	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0
0	0	0	0	0	1	1	0
0	0	0	0	0	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1

ACKNOWLEDGMENTS

First and foremost, I want to thank my advisor Dr. David Umphress from the computer science and software engineering department at Auburn University, Alabama. It has been an honor to work under him as his graduate research assistant. He helped me understand: the professional way of developing complex projects and the importance of project planning and project management. I appreciate all his contributions of time, ideas, vast knowledge, guidance, and funding to make my master's program productive and stimulating. His enthusiasm for innovative ideas and his unconditional support to venture into the world of unsolved problems has always been very motivational to me. I am thankful for the excellent example he provided as a successful individual and professor. His down-to-earth attitude towards the failures I experienced during the development of my thesis helped me stay positive through every failure I encountered, which eventually turned into successful research. He became my role model as a true leader.

I also want to thank Dr. Anthony Skjellum, who helped me realize my potential and put me on the path of being a researcher in the computer science field. He helped me get in contact with Dr. David Umphress. I am truly grateful to Dr. Bogdan Wilamowski's teachings and resources he provided to thoroughly understand the concept of neural networking.

I appreciate the support of my defense committee members: Dr. Bo Liu and Dr. Ahn Nguyen, for contributing their precious time to take an interest in my research, their insightful questions, and helpful advice.

I gratefully acknowledge my fellow collogues, my friends, and their contribution to my personal and professional time at Auburn University. I am especially grateful to Ash Searle, Heba Alwaneh, Fatma Neda, Fatima Hamade, Vaibhav Gupta, and Nirmit Patel, for being a source of friendships as well as a source of good advice and collaboration.

Lastly, I would like to thank my family for all their love and encouragement. I want to thank my father, Dr. Vibhutikumar Joshi, for his crucial financial support, moral encouragements, and insightful advice. Additionally, I want to thank my mother, Mrs. Nisha Joshi, for her unconditional love and support. I whole heartily and sincerely appreciate how my parents raised me; with love, encouragement of science, and their unlimited support in all of my pursuits.

Bhargav Vibhutikumar Joshi
Auburn University
Jun 2019

VITA

BHARGAV JOSHI
Cell: +1 (334)-332-7446
Email: bhargav.CSSE@gmail.com

Academic Record

- **Auburn University (MS with thesis), AL, USA** (Jan 2017 – Aug 2019)
  Computer Science and Software Engineering (Awarded Degree)
  Electrical Engineering (Extended studies done during Jan 2017 – Dec 2017)
- **Dharmsinh Desai University (B.Tech), India** (July 2012 – April 2016)
  Electronics and Communication Engineering

Work Experience

- **Auburn Cyber Research Center (ACRC), Auburn University, AL** (Jan 2018 – present)
  Position – Graduate Research Assistant
  Description – Working for Dr. David Umphress to research on cyber threats to PLC-based automation systems and SCADA/ICS systems. Worked on a project to secure PLC based systems using neural network.
- **Computer Science and Software Engineering, Auburn University, AL** (Jun 2017 – Jan 2018)
  Position – Graduate Research Assistant
  Description – Worked for Dr. Anthony Skjellum to explore cyber research field and introduce myself with industrial systems and industrial internet of things (IIOT)
- **Cotmac Electronics Pvt Ltd.**
  Position – Intern (Dec 2015 – Nov 2016)
  Description – Worked at this Company as an Intern to complete my **final semester project** in professional environment.

Major Academic Projects Accomplished (June 2014 – Aug 2019)

- **Implementation of Neural Network on PLC-based Automation Systems for Better Fault Tolerance and Error Detection**
  Implemented a neural network that was trained with a PLC's analog I/O values, digital I/O values, timer counts, and values of critical variables of ladder logic obtained during the simulation of PLC's code. The trained neural network can then generate predictions of ideal output values on the PLC's ports to monitor the system.
- **Speech Signal Processing using Microsoft Speech SDK**
  Speech signal processing algorithm developed by Microsoft speech SDK is used to detect spoken words in microphone and according to the detected word particular actions were executed by the program written in C# on Microsoft visual Studio platform.
- **Boiler Automation and Turbine Protection**
  A **Siemens S7-300 PLC** driven **boiler** which can very efficiently and automatically control the parameters of its operation is connected to a turbine. Sensors were designed and made compatible with PLC using Arduino.

Area of Interest

- Machine Learning
- Neural Network
- Software Process
- Embedded Systems